In Handbook of Evolutionary Machine Learning, W. Banzhaf, P. Machado, and M. Zhang (Eds.), New York, 2023. Springer.

Evolutionary Supervised Machine Learning

Risto Miikkulainen
The University of Texas at Austin and Cognizant AI Labs

This chapter provides an overview of evolutionary approaches to supervised learning. It starts with the definition and scope of the opportunity, and then reviews three main areas: evolving general neural network designs, evolving solutions that are explainable, and forming a synergy of evolutionary and gradient-based methods.

1 Introduction

In supervised learning problems, in principle the correct answers are known for each input. In the most general sense, the learner has access to a supervisor who tells what needs to be done for each input; in practice, such information is collected into a dataset ahead of time and sampled during the learning, and the dataset may be noisy and incomplete. The challenge in supervised learning is to construct a model that accurately predicts the outputs for new samples that are not in the dataset. Thus, the model imitates known behavior in a way that generalizes to new situations as well as possible (Figure 1).

Examples of supervised learning include classifying objects in visual images; diagnosing pathologies in X-ray images; predicting the next word in a stream of text; predicting future values of stocks; predicting the weather. Given that data is now collected routinely across various human activities in business, healthcare, government, education, and so on, many opportunities exist for using supervised learning in the real world.

The standard approach is gradient descent on neural networks, i.e. making small changes to the network weights (or parameters) in order to reduce the error (or loss) on known examples as quickly as possible. This approach has been commonplace since the 1980s (Schmidhuber, 2022, although conceived much earlier;). However, given a million-fold increase in computing power from just a couple of decades ago (Routley, 2017), it has recently become possible to scale supervised learning techniques to much larger datasets and architectures. Much of the power of modern supervised learning lies in this scale-up (Canatar et al., 2021). While the basis for this success still needs to be understood better, it has already revolutionized the field of artificial intelligence.

Evolutionary machine learning constitutes a distinctly different approach to supervised learning. It is not an incremental improvement method based on gradients, but instead a search method guided by fitness feedback, or reinforcement. It is therefore more general, but also cannot as naturally take advantage of massive datasets as gradient-based methods can. On the other hand, many domains in the real world do exist where such a scale-up is not possible. Datasets are sometimes very specific to the application (e.g. in a business domain, or medicine, or engineering design) and consist of thousands, instead of millions, of samples. In such domains, evolutionary supervised learning can provide two advantages, leading to two opportunities for advancing supervised machine learning.

The first opportunity is to take into account other goals than simply accuracy, i.e. to evolve general neural network designs. Note that a supervised signal (i.e. a gradient vector), can be converted to a fitness signal (i.e. a scalar, such as a norm of the gradient). Specific information will be lost; on the other hand, it is then possible to combine the accuracy goal with other goals. For instance, the networks evolved can be small

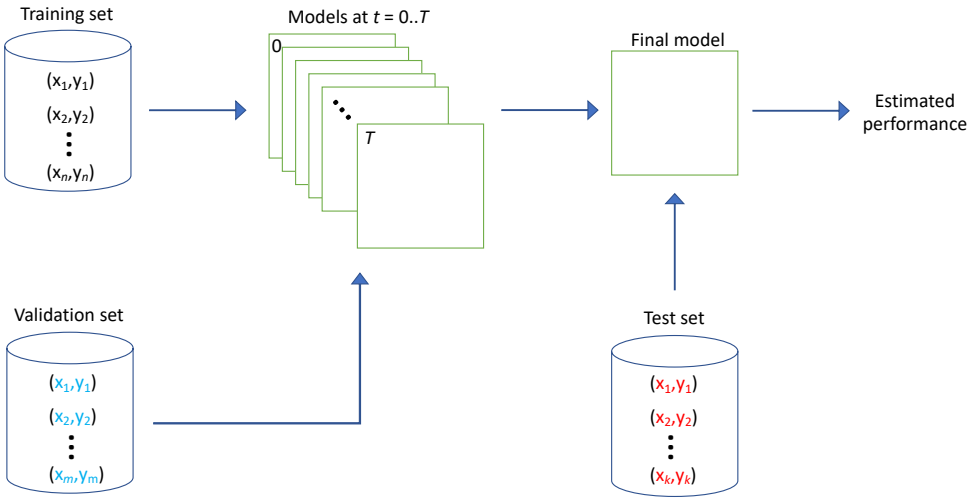


Figure 1: **A general setup for supervised learning.** Each input example x_i in the dataset is paired with the desired output y_i. The dataset is split into a training set that is used to modify the model over time, a validation set that is used to decide which model to choose as the final result of learning, and a test set that is used to estimate performance of the model on future examples. The system thus learns a model that imitates the known behavior and generalizes to future situations as well as possible.

enough to be transparent. It may be possible to improve regularization in such networks at the same time. Such networks can potentially take better advantage of minimal computing resources, and even fit better to hardware constraints.

The second opportunity is explainability. That is, gradients are inherently opaque, and machine learning systems based on them are essentially black boxes. This opaqueness makes it difficult to deploy them in many applications in the real-world. For instance, in domains like healthcare and finance it is important to be able to verify that the information used to draw the conclusion was relevant, unbiased, and trustworthy. In contrast, evolutionary machine learning can be used to discover solutions that are transparent. For instance, a solution can be represented in terms of decision trees, classifiers, programs, and rule sets.

A third opportunity for evolutionary supervised learning has emerged with the scale-up itself. Deep learning systems have become extremely large and complex, with many elements that interact nonlinearly. Like in many areas of system design, it is no longer possible for humans to design them optimally. However, evolutionary metalearning can be used to configure the architectures, hyperparameters, loss functions, activation functions, data selection and augmentation, and even learning methods so that the resulting deep learning models perform as well as possible. In this manner, it is possible to use evolution synergetically with gradient descent. Such automatic machine learning makes deep learning systems accessible to more people, and the designs can be optimized for size and data, making it possible to apply it more broadly.

These three opportunities will each be reviewed in the sections that follow.

2 Evolving general neural network designs

In evolving general neural network designs, gradient descent is replaced entirely with evolutionary optimization. Fitness is still at least partially based on the loss or accuracy on the supervised dataset, but only as a scalar value. Most importantly, evolution can be used to optimize other aspects of the design at the same time, such as explainability (e.g. through complexification), size (e.g. to improve regularization, or to fit within computational constraints), of hardware fit more generally (e.g. using threshold units or other non-differentiable components). These benefits currently apply to compact neural networks but could extend to deep learning as well.

2.1 Compact Neural Networks

Neural networks have been evolved for a long time, starting from the direct approaches of Montana and Davis (1989). Most of the techniques were developed for sequential decision tasks, especially those that are POMDP (Partially Observable Markov Decision Processes), i.e. tasks where the state is not fully observable and where recurrency thus matters. However, they can be used in supervised learning as well.

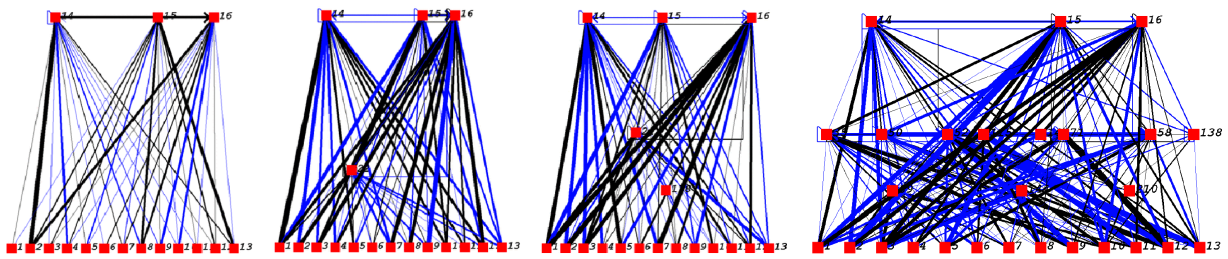
One example is the NEAT method (NeuroEvolution of Augmenting Topologies; Papavasileiou et al., 2021; Stanley and Miikkulainen, 2002), which optimizes both the architecture and the weights of a neural network. The main idea is to start small and complexify, which results in compact networks that are possible to understand. NEAT starts with a population of networks where the inputs are connected directly to the outputs. As evolution progresses, hidden nodes and recurrent as well as feedforward connections are added. Such architectural innovations are protected through speciation, i.e. they do not have to compete with other architectures until their parameters have been sufficiently optimized. Each innovation receives a unique identifier that allows taking advantage of crossover even among population of diverse architectures. As a result, NEAT evolves compact architectures where every complexification is evaluated and found useful or not.

Such compact networks are very different from deep learning networks in that they often employ principled and interpretable designs. For instance, in a combined foraging-pursuit-evasion task of simulated Khepera robots (Stanley and Miikkulainen, 2004), evolution first discovered a simple follow-the-opponent behavior that was often successful by chance: The agent occasionally crashed into the opponent when it had more energy than the opponent (Figure 2a). It then evolved a hidden node that allowed it to make an informed switch between behaviors: Attack when it had high energy, and rest when it did not (Figure 2b). Another added node made it possible to predict the agent's own and its opponent's energy usage from afar and attack only when victory was likely (Figure 2c). The most complex strategy, with several more nodes and complex recurrent connections between them, allowed the agent to predict also the opponent's behavior, encourage it to make mistakes, and take advantage of the mistakes to win (Figure 2d).

Although this example illustrates a sequential decision task, the same principles apply to supervised learning with NEAT. Note that these principles are very different from those in deep learning. Performance with very large networks is based on overparameterization where individual components perform only minimal operations: for instance, the residual module in ResNet architectures combines bypassing the module with the transformation that the module itself computes (He et al., 2016). In contrast, in NEAT every complexification is there for a purpose that can in principle be identified in the evolutionary history. It thus offers an alternative solution, one that is based on principled neural network design.

This kind of compact evolved neural networks can be useful in four ways in supervised learning:

1. They can provide an explainable neural network solution. That is, the neural network still performs



(a) (b) (c) (d)

Figure 2: **Evolutionary Discovery through Complexification.** As the NEAT neuroevolution method (Stanley and Miikkulainen, 2002) complexifies networks, the behaviors that these networks generate become more complex as well. It is thus possible to identify how the network generates the behavior it does. In the simultaneous pursuit-evasion-foraging task for simulated Khepera robots, the approach discovered (a) opponent following, (b) behavior selection, (c) opponent modeling, and (d) opponent control through several such complexification steps (Stanley and Miikkulainen, 2004). Nodes are depicted as red squares and numbered in the order they were created. Positive connections are black and negative are blue, recurrent connections are indicated by triangles, and the width of the connection is proportional to its strength. Complexifying evolution thus provides a way of understanding network performance. (Figures from Stanley, 2004)

based on recurrency and embeddings, but its elements are constructed to provide a particular functionality, and therefore its behavior is transparent (Aharonov-Barki et al., 2001; Ijspeert, 2008; Kashtan and Alon, 2005; Stanley and Miikkulainen, 2004).

2. They can provide regularized neural network solutions, instead of overfitting to the dataset. The networks are compact, which generally regularization (Ganon et al., 2003; Oymak, 2018; Reed, 1993), and they are chosen based on their overall performance instead of fine-tuned to fit individual examples. This property should be particularly useful when the datasets are relatively small, which is the case in many practical applications. Thus they can extend the scope of machine learning applications.

3. They can utilize minimal hardware resources well. The advantages of deep-learning networks do not emerge until a very large number of parameters. If the hardware does not allow that scale (e.g. in edge devices), evolved networks provide an alternative principle that can be optimized to the given resources.

4. They can be constructed to fit hardware constraints. Gradient descent in principle requires high precision weights and differentiable activation functions that are expensive to implement in hardware. In contrast, evolution can be used to optimize the performance of networks with e.g. quantized weights, linear threshold units, or FPGA-compatible components that are easier to implement (Gaier and Ha, 2019; Liu et al., 2021b; Shayani et al., 2008).

While examples of each of these advantages exist already, their large-scale applications are still future work and an interesting research opportunity. Another interesting opportunity is to scale them up to large networks, discussed next.

2.2 Deep Networks

As discussed above, evolution of entire networks usually focuses on small, recurrent architectures with up to hundreds of thousands of parameters. Evolutionary operations can still search such a space effectively. At the

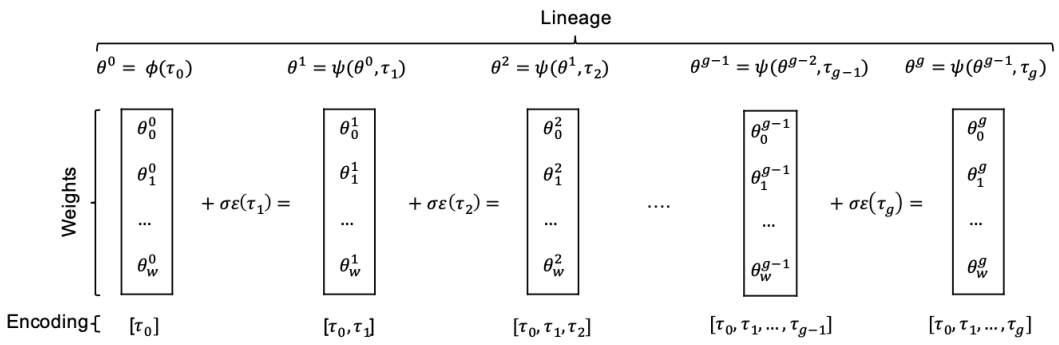


Figure 3: **Compact Indirect Encoding of Deep Neural Networks.** In the approach of Such et al. (2017), a population of initial networks with parameters θ^0 are first created through an initialization function ϕ. Subsequent generations are created through mutations ψ seeded τ_i. Only the seeds τ_i need to be stored and evolved; the current population can be created by applying the same sequence of mutations to θ^0. In this way, indirect encoding makes it possible to represent deep neural networks compactly enough so that they can be evolved in their entirety. (Figure from Such et al., 2017)

outset, it seems difficult to scale this approach to deep learning networks, which routinely contain hundreds of millions of parameters, and sometimes up to hundreds of billions of them.

There are special techniques that may make it possible to optimize solutions in such very high-dimensional spaces. For instance, Deb and Myburgh (2017) showed that in the specific case of metal casting scheduling, it was possible to establish a constrained search, and solve problems with up to a billion parameters. It may be possible to develop similar techniques for neuroevolution as well and thereby scale up to deep learning networks.

Another approach is to utilize indirect encodings. Instead of optimizing each parameter independently, a coding is evolved that is then mapped to the final design through a developmental or a generative process. For instance in the HyperNEAT approach (Stanley et al., 2009), such a decoding neural network is evolved to output weight values for another neural network that actually performs the task. The performance network is embedded into a substrate where each of its neurons is located at particular coordinates; given the coordinates of the source and target neuron as input, the decoding neural network generates the weight of the connection between them as its output. Importantly, it is possible to sample the substrate at different resolutions, i.e. embed a very large number of neurons in it, and thus use the same decoding network to generate very large performance networks. The decoding networks are termed compositional pattern-producing networks because the weights they generate often follow regular patterns over the substrate. This observation suggests that the HyperNEAT approach could be used to generate structures over space, such as 2D visual or 1D time series domains. Whether it can be extended to layered structures and general architectures remains an open question

Evolving such structured deep learning networks may be achieved by other means. For instance, Such et al. (2017) developed a method where each individual network is represented indirectly as a sequence of mutation operators performed on an initial set of weight parameters (Figure 3). The initial set may have millions of parameters, but they only need to be stored once for each individual lineage. The mutations can be encoded efficiently as a sequence of seeds that generate the changes through a precomputed table. Thus, the compression rate depends on the number of generations but is in the order of $10^3 - 10^4$-fold. It thus makes it possible to evolve deep learning networks with millions of parameters, as was demonstrated in several tasks in the Atari game-playing domain.

While evolution of entire very large networks is still a new area, these initial results suggest that it may

indeed be possible. The benefits of general design, reviewed in the previous section, may then apply to them as well.

3 Evolving explainable solutions

Most AI today is based on neural networks. Given the vast amount of available data and compute, impressive performance is indeed possible in many real-world classification and prediction tasks. However, neural networks are inherently opaque. The knowledge they have learned is embedded in very high-dimensional vector spaces, with a lot of redundancy and overlap, and with nonlinear interactions between elements. Even though they may perform well on average, it is difficult to deploy such a system when we do not understand how it arrives at a particular answer in any individual case. The system may be overfitting, fooled by adversarial input, missing an important factor, or utilizing unfair bias to make its decision (Dai et al., 2020; Huang et al., 2020; Sharma et al., 2020). Explainability has thus emerged as one of the main challenges in taking AI to the real world.

Evolving compact neural networks through complexification may make their function transparent. However, explainability in supervised learning requires more: For each new example, the system should be able to identify the information and principles used to arrive at the decision. Structures different from neural networks are thus needed, such as decision trees, classifiers, programs, and rules. While they cannot be easily modified by gradients, evolutionary machine learning can be used to construct them, as reviewed in this section.

3.1 Decision trees

Decision trees are a classical supervised learning method with low computational overhead and transparent, explainable performance (Breiman et al., 1984; Quinlan, 1986, 1993). In its most basic form, samples are represented as vectors of features, and the tree is used to classify them into discrete categories. Each node of the tree looks at one feature, and based on its value, passes the sample down to one of the nodes below it; the leaf nodes each assign a category label.

The traditional approach to constructing a decision tree is to do it top down in a greedy fashion: At each step, a feature that is deemed most useful for branching is chosen, and the process continues recursively in each branch until all training samples in a branch have the same category label. The choice can be based e.g. on the information gain in the branching, which is a heuristic aimed at constructing simple and shallow trees, which in turn are likely to generalize better.

However, the greedy construction is often suboptimal. It is difficult to take into account interactions between features, and construct trees that are balanced, and often trees end up complex and prone to overfitting (Barros et al., 2012). Evolutionary search provides an alternative: it can search for optimal trees globally, taking into account both accuracy and size objectives (Barros et al., 2012).

A natural approach to evolving decision trees is to represent them, indeed, as trees where the nodes specify a comparison of a particular feature with a constant value, or a label if the node is a leaf (Aitkenhead, 2008). Fitness can then consist of accuracy in the training set, but also include complexity of the tree, such as depth and balance. Such approaches achieve competitive performance e.g. in the UCI benchmarks, especially in tasks that are prone to overfitting (Jankowski and Jackowski, 2014).

One way to improve generalization with decision trees is to form ensembles of them, including methods such as random forests, where different trees are trained with different parts of the dataset (Breiman, 2001).

Evolutionary optimization can be applied to such ensembles as well: The nodes of the tree (represented e.g. as feature+value pairs) can be concatenated into a vector, and multiple such vectors representing the entire ensemble can then be used as an individual in the population. The resulting evolutionary ensembles can outperform e.g. the standard random forests and AdaBoost methods of forming decision tree ensembles (Dolotov and Zolotykh, 2020).

Thus, evolutionary optimization provides a potentially powerful way to overcome some of the shortcomings of decision trees, thus having a large impact in domains with few examples, meaningful features, and the need for transparent decision-making.

3.2 Learning classifier systems

The two classic evolutionary machine learning approaches, learning classifier systems (LCS) and genetic programming (GP), can be applied to supervised learning tasks as well. They may perform well on specific domains especially when there is too little data to take advantage of deep learning. However, like decision trees, their solution structure is transparent and they can thus be used to construct explainable solutions.

LCS has a long history of development, and includes many variations (Butz et al., 2008; De Jong, 1988; Holland, 1986; Urbanowicz and Moore, 2009). The approach was originally developed for reinforcement learning problems, and a more thorough overview of it will be given in that context in Chapter 4. Often a distinction is made between the Michigan-style LCS, where the population consists of individual classifiers and the solution of the entire population, and Pittsburgh-style LCS, where the population consists of sets of classifiers and the solution is the best such set. Pittsburgh-style LCS is often referred to as ruleset evolution, and will be discussed in Section 3.4 below.

In Michigan-style LCS, the population consists of individual IF-THEN rules whose antecedents specify feature values (0, 1, don't_care) in the environment, and consequence is a binary classification (0, 1). In a supervised setting, the rules that match an input example are assigned fitness based on whether their classification is correct or not. The fitness is accumulated over the entire training set, and fitness then used as the basis for parent selection. Offspring rules are generated through crossover and mutation as usual in a genetic algorithm. The population can be initialized to match training examples, and it is grown incrementally, which makes learning and performance easier to understand.

For instance, LCS methods of supervised learning have been developed for for data mining in noisy, complex problems such as those in bioinformatics Urbanowicz et al. (2014). Expert knowledge can be encoded as probability weights and attribute tracking, guiding search towards more likely solutions. Applied to a large number of single-nucleotide polymorphism datasets, the approach proved to be effect across a range of epistasis and heterogeneity.

The final set of LCS rules is transparent, although it can grow very large and difficult to interpret. Methods for compacting and filtering it have been developed to improve interpretability. However, Pittsburgh-style rule evolution often results in smaller and more interpretable rule sets, as will be discussed in Section 3.4 below.

3.3 Genetic Programming

Along the same lines, GP is primarily a method for reinforcement learning domains, and will be reviewed in detail in that context in Chapter 14. The main idea in GP is to evolve programs, usually represented by trees of elementary operations Langdon et al. (2008). This idea is very general, and also applies to the supervised learning context. The program can represent a function that transfers inputs to outputs. Those inputs and

outputs can originate from a supervised dataset, and fitness for a program measured based on how often the output labels are correct. GP can thus evolve programs that perform well in the supervised task.

The idea is that domain knowledge about the features and operations can be encoded in the search space of the program. While in deep learning approaches, such knowledge is usually extract from the examples, GP only needs to find how to use it. Therefore, it is possible to learn from much fewer examples. Also the programs are transparent, and therefore it is possible to explain how the system arrives at an answer.

For instance, in the EDLGP system (evolutionary deep learning GP; Bi et al., 2022), GP was used to classify images in standard object datasets CIFAR-10, SVHN, FashionMNIST, and two face image datasets. Starting from images as input, the programs consisted of several layers of processing, including image filtering, feature extraction, concatenation, and classification. At each level, a number of operators were provided as elements, and the goal was to find a tree that utilizes these operators at the appropriate levels most effectively. Thus, much knowledge about image processing was provided to the search, making it possible to construct effective functions with only a few training examples. Indeed, the best evolved trees performed much better than standard deep learning approaches such as CNN and ResNet when only one to eight training examples were available for each class; when 10-128 were available, they two approaches were roughly comparable.

Moreover, the resulting trees are transparent, taking advantage of operations that make sense in the domain (Figure 4. Thus, the example demonstrates once again that evolutionary supervised learning makes sense in low-data environments in general, and also provides a level of explainability that is missing from the standard deep learning approaches.

3.4 Rulesets

Compact neural networks, decision trees, classifier populations, and programs may be transparent, but the most natural and clear form of explainability is in terms of rules. Indeed, AI has a rich tradition of rule-based systems Hayes-Roth (1985). Within their scope, they provide a rigorous inference mechanism based on logic. Importantly, they also provide explainability in that the rules specify exactly what knowledge is used to arrive at a decision, and how the decision is made. In other words, they provide precisely what is missing from current neural-network-based AI.

On the other hand, rule-based systems are usually constructed by hand, to encode knowledge of human experts, instead of learning such knowledge from supervised datasets. Rule-based systems are thus an important opportunity for evolutionary machine learning. Whereas rules cannot be learned through gradient descent, they can be learned through evolution.

First, rules are represented in a structure that can be evolved with e.g. crossover and mutation operators. For instance, an individual in a population can consist of a set of rules, each with a left-hand side that consists of logical clauses and/or arithmetic expressions based on domain features, and a right-hand side that specifies a classification or prediction (Shahrzad et al., 2022; Srinivasan and Ramakrishnan, 2011). Then, the number of rules, their order, the number of features and the features themselves, the logical and arithmetic operations between them, and coefficients are evolved, as are the actions and their coefficients on the right-hand side. Furthermore, there can be probabilities or confidence values associated with each rule, and ways of combining outputs of multiple rules can be evolved as well.

In other words, rule-set evolution can take advantage of existing representations in Turing-complete rule-based systems. As usual in supervised learning, the fitness comes from loss, or accuracy, across the dataset. Instead of coding the rule-based system by hand, the entire system can then be evolved. The result, in principle, is explainable machine learning.

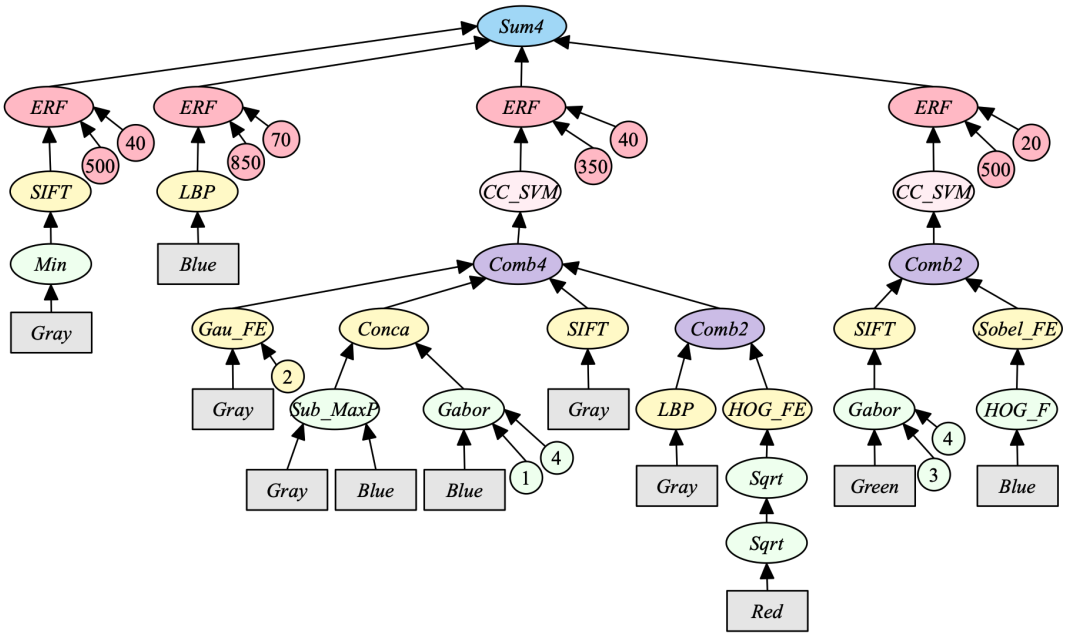


Figure 4: **An image classification tree discovered by GP.** The tree was evolved with a training set of 80 images per class in CIFAR-10, reaching accuracy of 52.26%, which is better than deep learning approaches Bi et al. (2022). The gray nodes at the bottom indicate input channels; green nodes perform filtering operations; yellow nodes extract features; purple nodes form combinations of features; red modes make classification decisions. taking advantage of image processing knowledge given in terms of layers and elements, the operations are chosen and the tree structure formed through GP. It results in effective classification even with very few examples, and the classification decisions are transparent. (Figure from Bi et al., 2022)

This general approach has already been demonstrated in several decision-making tasks, including stock trading, game playing, robotic control, and diabetes treatment recommendation (Shahrzad et al., 2020, 2022). However, it has also been applied to the supervised task of blood-pressure prediction in intensive care (Hemberg et al., 2014; Shahrzad et al., 2022, Figure 5;). Based on a time series of blood-pressure measurements, that task was to predict whether the patient will go into a septic shock in the next 30 minutes, while there is still time to prevent it. Indeed, the evolved rulesets achieved an accuracy of 0.895 risk-weighted error on the unseen set, with a true positive rate of 0.96 and a false positive rate of 0.39 (Figure 5). Most importantly, the rules were evaluated by emergency-room physicians who found them meaningful. Without such explicit rules, it would be very hard to deploy the prediction system. With rules, human experts can check the AI's reasoning and decide whether to trust it, thus making the AI helpful as a first indicator of potential problems. Evolution of rulesets in this supervised task makes it possible.

4 Evolutionary Metalearning

The idea of metalearning is to enhance the general setup of the supervised learning system through a separate learning process. The two systems thus work synergetically; for instance, the fitness of an evolved neural network design is based on the performance of supervised training of that network. Metalearning is useful, and even necessary, because modern learning systems have become too large and complex for humans to optimize (Elsken et al., 2019; Hoos, 2012; Liang and Miikkulainen, 2015; Sinha et al., 2014). There are

1.(*Mean*[4] $<$ 72.75mmHg) & (*Kurtosis*[3] $<$ 4.09) $\longrightarrow$*Low*
2.(*Skew*[10] $>$ 2.01) & (*Mean*[8] $<$ 88.92mmHg) & (*Skew*[4] $<$ 0.15) $\longrightarrow$*Normal*
3.(*Mean*[0] $<$ 72.75mmHg) $\longrightarrow$*Low*
4.(*Mean*[10] $<$ 73.10mmHg) $\longrightarrow$*Low*
5.(*Mean*[1] $<$ 121.96mmHg) & (*Mean*[4] $>$ 88.92mmHg) & (*Mean*[1] $>$ 73.10mmHg) $\longrightarrow$*High*
6.(*Mean*[0] $<$ 97.53mmHg) $\longrightarrow$*Normal*
7.(*Mean*[0] $<$ 97.53mmHg) & (*Kurtosis*[0] $>$ 12.71) $\longrightarrow$*Normal*
8.(*Mean*[4] $<$ 72.75mmHg) & (*Kurtosis*[7] $>$ 4.03) $\longrightarrow$*Low*
9.(*Mean*[4] $>$ 121.96mmHg) & (*Kurtosis*[5] $>$ 12.71) & (*Kurtosis*[3] $>$ 1.00) $\longrightarrow$*Normal*
10.(*Std*[0] $<$ 10.76) $\longrightarrow$*High*
11.(*Kurtosis*[0] $>$ 1.00) $\longrightarrow$*High*
12.(*Mean*[0] $<$ 72.75mmHg) & (*Std*[4] $>$ 0.01) $\longrightarrow$*Low*
13.(*Kurtosis*[0] $<$ 4.09) & *Skew*[3] $>$ 2.01) $\longrightarrow$*Normal*
14.(*Skew*[9] $>$ 0.06) $\longrightarrow$*High*
15.(*Skew*[0] $<$ 1.95) $\longrightarrow$*High*
16.(*Mean*[0] $<$ 72.75mmHg) & (*Mean*[5] $<$ 52.12mmHg) $\longrightarrow$*Low*
17. Default $\longrightarrow$*Normal*

Figure 5: **A transparent and explainable rule set evolved to predict blood pressure.** EVOTER discovered sets of features at specific time points to provide a useful signal for prediction. For instance, *Std*[4] specifies the standard deviation of the aggregated mean arterial pressure (MAP) over four minutes earlier. The evolved rules predict sepsis within 30 mins with a 0.895 risk-weighted error on the unseen set, with a true positive rate of 0.96 and a false positive rate of 0.394. Most importantly, the rules are interpretable and meaningful to experts, which makes it much easier to deploy in practice compared to e.g. black-box neural network models. (Figure from Shahrzad et al., 2022)

many dimensions of design, such as network topology, components, modularity, size, activation function, loss function, learning rate and other learning parameters, data augmentation, and data selection. These dimensions often interact nonlinearly, making it very hard to find the best configurations.

There are many approaches to metalearning, including gradient-based, reinforcement learning, and Bayesian optimization methods (Elsken et al., 2019; Schaul and Schmidhuber, 2010). They can be fast and powerful especially within limited search spaces. However, the evolutionary approach is most creative and versatile (Liang et al., 2019; Liu et al., 2021a; Miikkulainen et al., 2021). It can be used to optimize many of the design aspects mentioned above, including those that focus on optimizing discrete configurations that are less natural for the other methods. It can also be used to achieve multiple goals: It can be harnessed to improve state-of-the-art performance, but also to achieve good performance with little expertise, fit the architecture to hardware constraints, take advantage of small datasets, improve regularization, find general design principles as well as customizations to specific problems. There is also an incipient and future opportunity to find synergies between multiple aspects of optimization and learning.

This section reviews the various aspects of evolutionary metalearning. Interestingly, many of the techniques originally developed for evolving entire networks (e.g. those in Section 2.1) have a new instantiation as metalearning techniques, evolving network architectures synergetically with learning. In addition, new techniques have been developed as well, especially for other aspects of learning system design.

4.1 Neural Architecture Search

The most well-studied aspect of metalearning is Neural Architecture Search (NAS), where the network wiring, including potentially overall topology, types of components, modular structure and modules themselves, and general hyperparameters are optimized to maximize performance and other metrics (Elsken et al., 2019; Liu et al., 2021a). NAS in general is a productive area in that many approaches have been developed and they work well. However, even random search works well, suggesting that the success has less to do with the methods so far but rather with the fact that architectures matter and many good designs are possible. It also suggests that further improvements are still likely in this area—there is no one dominant approach or solution.

One of the early and most versatile approaches is CoDeepNEAT (Liang et al., 2019; Miikkulainen et al., 2023). This approach builds on several aspects of techniques developed earlier for evolving complete networks. In SANE, ESP, and CoSYNE, partial solutions such as neurons and connections were evolved in separate subpopulations that were then combined into full solutions, i.e. complete neural networks, with the global structure specified e.g. in terms of a network blueprint that was also evolved (Gomez and Miikkulainen, 1997; Gomez et al., 2008; Moriarty and Miikkulainen, 1997). Similarly, CoDeepNEAT co-evolves multiple populations of modules and a population of blueprints that specifies which modules are used and how they are connected into a full network (Figure 6*a*). Modules are randomly selected from the specified module population to fill in locations in the blueprint. Each blueprint is instantiated in this way many times, evaluating how well the design performs with the current set of blueprints. Each module participates in instantiations of many blueprints (and inherits the fitness of the entire instantiation each time), thus evaluating how well the module works in general with other modules. The main idea of CoDeepNEAT is thus to take advantage of (and scale up with) modular structure, similarly to many current deep learning designs such as the inception network and the residual network (He et al., 2016; Szegedy et al., 2015).

The modules and the blueprints are evolved using NEAT (Section 2.1), again originally designed to evolve complete networks and adapted in CoDeepNEAT to evolving network structure. NEAT starts with a population of simple structures connecting inputs straight to outputs, and gradually adds more modules in the middle, as well as parallel and recurrent pathways between them. It thus prefers simple solutions but complexifies the module and blueprint structures over time as necessary. It can in principle design rather complex and general network topologies. However, while NEAT can be used to create entire architectures directly, in CoDeepNEAT it is embedded into the general framework of module and blueprint evolution; it is thus possible to scale up through repetition that would not arise from NEAT naturally.

The power of CoDeepNEAT was originally demonstrated in the task of image captioning, a domain where a competition had been run for several years on a known dataset (Miikkulainen et al., 2023). The best human design at that point, the Show&Tell network (Vinyals et al., 2015), was used to define the search space; that is, CoDeepNEAT was set to find good architectures using the same elements as in the Show&Tell network. Remarkably, CoDeepNEAT was able to improve the performance further by 15%, thus demonstrating the power of metalearning over best human solutions (Miikkulainen et al., 2023). Interestingly, the best networks utilized a principle different from human-designed networks: They included multiple parallel paths, possibly encoding different hypotheses brought together in the end (Figure 6*b*). In this manner, the large search space utilized by CoDeepNEAT may make it possible to discover new principles of good performance.

Similar CoDeepNEAT evolution from a generic starting point has since then been used to achieve a state-of-the-art in text classification (Wikidetox; Liang et al., 2019) and image classification (Chest X-rays; Liang et al., 2019). The experiments also demonstrated that with very little computational cost it is possible to achieve performance that exceeds standard architectures, making it possible to quickly and effectively deploy deep learning to new domains. CoDeepNEAT has since then been extended with mechanisms for multiobjective optimization, demonstrating that the size of the network can be minimized at the same time

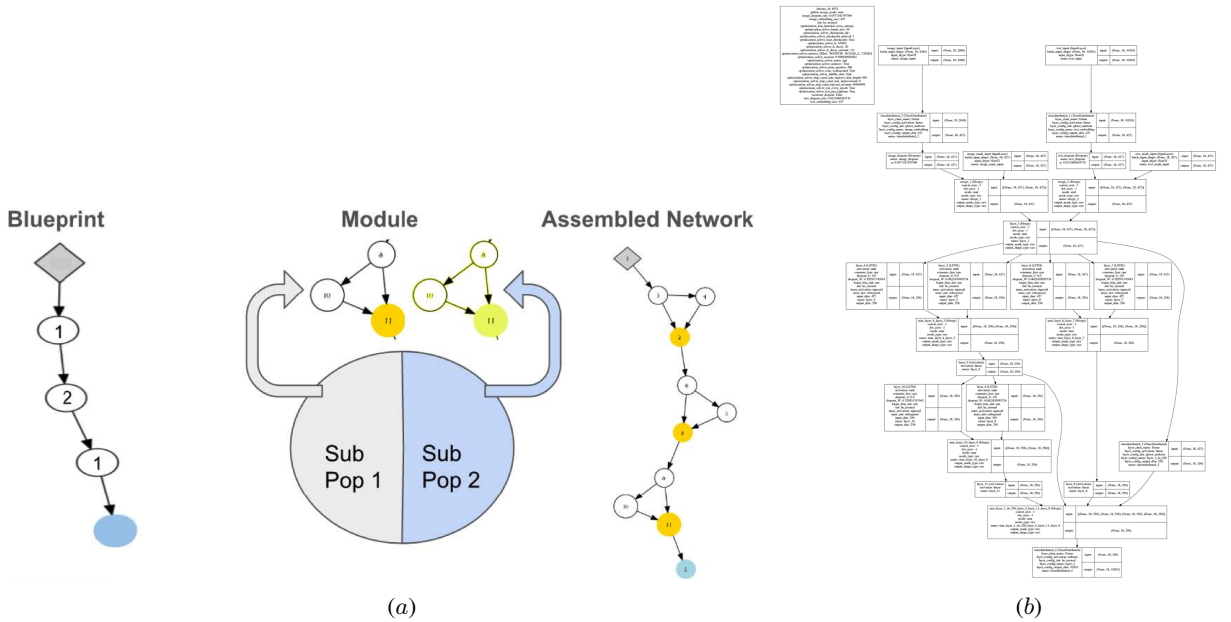


Figure 6: **Discovering Complex Neural Architectures through Coevolution of Modules and Blueprints.** (a) In CoDeepNEAT (Miikkulainen et al., 2023), the blueprints represent the high-level organization of the network and modules fill in its details. The blueprint and module subpopulations are evolved simultaneously based on how well the entire assembled network performs in the task. This principle was originally developed for evolving entire networks (Gomez and Miikkulainen, 1997; Moriarty and Miikkulainen, 1997), but it applies in neural architecture search for deep learning as well. (b) The overall structure of a network evolved for the image captioning task; the rectangles represent layers, with hyperparameters specified inside each rectangle. One module consisting of two LSTM layers merged by a sum is repeated three times in the middle of the network. The approach allows discovery of a wide range of network structures. They may take advantage of principles different from those engineered by humans, such as multiple parallel paths brought together in the end in this network. (Figures from Miikkulainen et al., 2023)

as its performance. Indeed, size can sometimes be minimized to 1/12 with only a small (0.38%) cost in performance (Liang et al., 2019). Another useful extension is to multitask learning by incorporating task routing, i.e. coevolution of task-specific topologies instead of a single blueprint, all constructed from the same module subpopulations (Liang et al., 2018). In multitask learning, it is possible to learn accurate performance even with small datasets. Simultaneous learning of multiple datasets utilizes synergies between them, resulting in performance that exceeds learning in each task alone. Multitask evolution finds architectures that best support such synergies. The approach achieved state-of-the-art in e.g. multialphabet character recognition (Omniglot; Liang et al., 2018), as well as multiattribute face classification (CelebA; Meyerson and Miikkulainen, 2018). These extensions again make it possible to apply deep learning to domains where hardware or data is limited, as it often is in real-world applications.

An important question about evolutionary NAS, and about metalearning in general, is whether it really makes a difference, i.e. results in architectures that advance the state of the art. Such an argument is indeed made below in Section 4.2.5 wrt. synergies of different aspects of metalearning. However, perhaps a most convincing case wrt. NAS is the AmoabaNet (Real et al., 2019). At its time, it improved the state-of-the-art in the ImageNet domain, which had been the focus of deep learning research for several years.

There were three innovations that made this result possible. First, search was limited to a NASNet search space, i.e. networks with a fixed outer structure consisting of a stack of inception-like modules (Figure 7a).

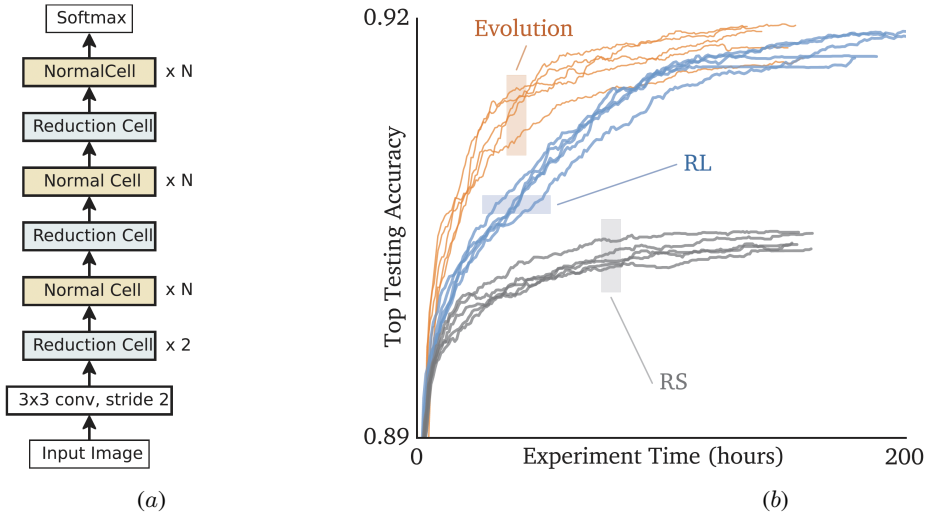


Figure 7: **Evolutionary Discovery in the NASNet Search Space Compared RL and Random Search.** (a) The AmoebaNet method (Real et al., 2019) focuses evolution to a particular stacked architecture of inception-like normal and reduction modules (cells); these networks are then scaled to larger sizes algorithmically. AmoebaNet also promotes regularization by removing the oldest individuals in the population. (b) As a result, it discovers architectures that are more accurate than those discovered through random search and RL, reaching state-of-the-art accuracy in standard benchmarks like ImageNet. (Figures from Real et al., 2019)

There were two different module architectures, normal and reduction; they alternate in the stack, and are connected directly and through skip connections. The architecture of the modules is evolved, and consists of five levels of convolution and pooling operations. The idea is that NASNet represents a space of powerful image classifiers that can be searched efficiently. Second, a mechanism was devised that allowed scaling the architectures to much larger numbers of parameters, by scaling the size of the stack and the number of filters in the convolution operators. The idea is to discover good modules first and then increase performance by scaling up. Third, the evolutionary process was modified to favor younger genotypes, by removing those individuals that were evaluated the earliest from the population at each tournament selection. The idea is to allow evolution to explore more instead of focusing on a small number of genotypes early on. Each of these ideas is useful in general in evolutionary ML, not just as part of the AmoebaNet system.

Indeed, AmoebaNet's accuracy was the state of the art in the ImageNet benchmark at the time, which is remarkable given how much effort the entire AI community had spent on this benchmark. Experiments also demonstrated that evolutionary search in NASNet was more powerful than reinforcement learning and random search in CIFAR-10, resulting in faster learning, more accurate final architectures, and ones with lower computational cost (Figure 7b). It also demonstrates the value of focusing the search space intelligently so that good solutions are in that space, yet it is not too large to find them.

Evolutionary NAS is rapidly becoming a field of its own, with several new approaches proposed recently. They include multiobjective and surrogate-based approaches (Lu et al., 2020), Cartesian genetic programming (Suganuma et al., 2020; Wu et al., 2021), variable-length encodings and smart initialization (Sun et al., 2020a,b), and LSTM design (Rawal and Miikkulainen, 2020), to name a few; see Chapter X for a more detailed review. The work is expanding from convolutional networks to transformers and diffusion networks, and from visual and language domains to tabular data. An important direction is also to optimize design aspects other than the architecture, as will be reviewed next.

4.2 Beyond Architecture Search

Besides the architecture, there are several other aspects of machine learning system design that need to be configured properly for the system to perform well. Those include learning hyperparameters (such as the learning rate), activation functions, loss functions, data sampling and augmentation, and the methods themselves. Approaches similar to those used in NAS can be applied to them; however, the evolutionary approach has an advantage in that it is the most versatile: It can be applied to graphs, vectors of continuous and discrete parameters, and configuration choices. This ability is particularly useful as new architectures are developed. For instance, at this writing, work has barely begun on optimizing designs of transformer (Vaswani et al., 2017) or diffusion (Sohl-Dickstein et al., 2015) architectures. They have elements such as attention modules, spatial embeddings, and noise transformations that are different from prior architectures, yet may be parameterized and evolution applied to optimize their implementation. Most importantly, evolution can be used to optimize many different aspects of the design at the same time, discovering and taking advantage of synergies between them. Several such approaches are reviewed in this section.

4.2.1 Loss functions

Perhaps the most fundamental is the design of a good loss function. The mean-squared-error (MSE) loss has been used for a long time, and more recently the cross-entropy (CE) loss has become popular, especially in classification tasks. Both of those assign minimal loss to outputs that are close to correct, and superlinearly larger losses to outputs further away from correct values. They make sense intuitively and work reliably, so much so that alternatives are not usually even considered.

However, it turns out that it is possible to improve upon them, in a surprising way that would have been difficult to discover if evolution had not done it for us (Gonzalez and Miikkulainen, 2020, 2021b). If outputs that are extremely close to correct are penalized with a larger loss, the system learns to avoid such extreme outputs—which minimizes overfitting (Figure 8a). Such loss functions, called Baikal loss for their shape, lead to automatic regularization. Regularization in turn leads to more accurate performance on unseen examples, especially in domains where the amount of available data is limited, as is the case in many real-world applications.

Baikal loss was originally discovered with a classic genetic programming approach where the function was represented as a tree of mathematical operations (Gonzalez and Miikkulainen, 2020). The structure of the tree was evolved with genetic algorithms, and the coefficients in the nodes with CMA-ES (Hansen and Ostermeier, 2001). This approach is general and creative in that it can be used to explore a large search space of diverse functions. However, many of those functions do not work well and often are not even stable. In the follow-up TaylorGLO method (Gonzalez and Miikkulainen, 2021b), the functions were represented instead as third-order Taylor polynomials. Such functions are continuous and can be directly optimized with CMA-ES, making the search more effective.

Regularization in general is an important aspect of neural network design, there are many techniques available, such as dropout, weight decay, and label smoothing (Hanson and Pratt, 1988; Srivastava et al., 2014; Szegedy et al., 2016), but how they work is not well understood. Loss-function optimization, however, can be understood theoretically, and thus provides a starting point to understanding regularization in general (Gonzalez and Miikkulainen, 2021a). It can be described as a balance of two processes, one a pull toward the training targets, and another a push away from overfitting. The theory leads to a practical condition for guiding the search toward trainable functions.

Note that Baikal loss is a general principle; evolutionary optimization was crucial in discovering it but it can now be used on its own in deep learning. It is still possible to customize it for each task and architecture,

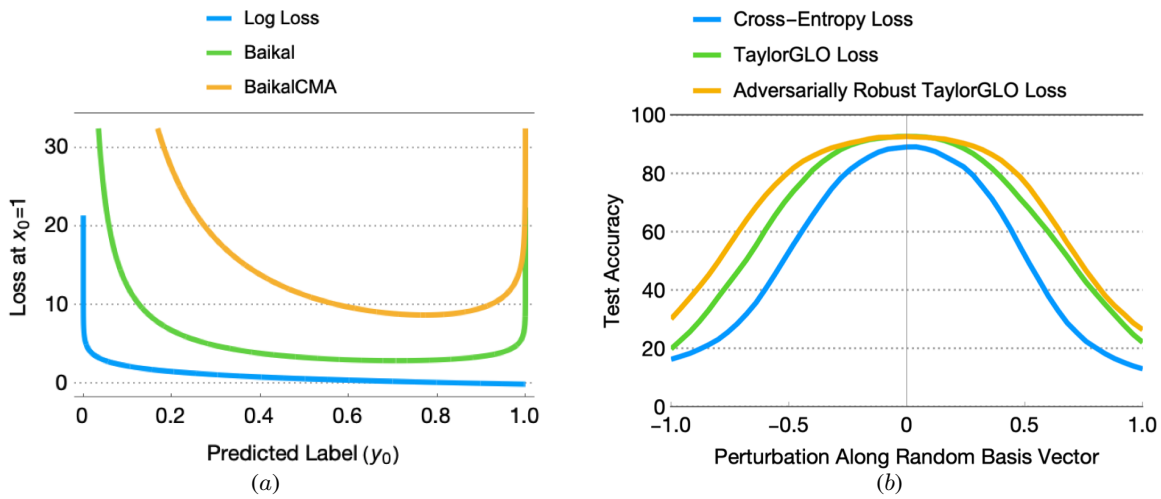


Figure 8: **Regularization and Robustness with Evolved Loss Functions.** (a) The standard loss function, such as Log Loss (or Cross-Entropy) has a high loss for outputs that are far from correct (1.0 in this case) and a low loss otherwise. In contrast, evolutionary optimization of loss functions through GLO/TaylorGLO (Gonzalez and Miikkulainen, 2020, 2021b) discovered a new principle: When the output is very close to the correct one, a high loss is incurred. This principle, termed Baikal loss for its shape, discourages overfitting, thus regularizing the network automatically, leading to better generalization. Such a loss is effective but counterintuitive, and thus unlikely to be discovered by human designers. (b) The Baikal loss also makes the network performance more robust. This effect can be quantified by perturbing the network weights. With Baikal loss, the network's performance is less affected than with Cross-Entropy loss. This effect can be further magnified by making robustness against adversarial inputs an explicit second objective in evolution. Thus, loss-function optimization can be used to improve not just regularization, but robustness as well. (Figures from Gonzalez and Miikkulainen, 2020, 2021a)

and even small modifications to the standard Baikal shape may make a difference. Optimization may also have a significant effect on various learning challenges, for instance when there is not much training data (Gonzalez et al., 2019), or when the labels are particularly noisy (Gao et al., 2021). It may also be possible to modify the loss function during the course of learning, for instance by emphasizing regularization in the beginning and precision towards the end (similarly to activation functions; Section 4.2.2).

It turns out that loss functions that regularize also make networks more robust, and this effect can be further enhanced by including an explicit robustness goal in evolution (Figure 8b). One way to create such a goal is to evaluate performance separately wrt. adversarial examples. This result in turn suggests that loss-function optimization could be an effective approach to creating machine learning systems that are robust against adversarial attacks.

Loss-function optimization can also play a major role in systems where multiple loss functions interact, such as Generative Adversarial Networks (GANs; (Gonzalez et al., 2023)). GANs include three different losses: discriminative loss for real examples and for fake examples, and the generative loss (for fake examples). It is difficult to get them right, and many proposals exist, including those in minimax, nonsaturating, Wasserstein, and least-squares GANs (Arjovsky et al., 2017; Goodfellow et al., 2014; Mao et al., 2017). Training often fails, resulting e.g. in mode collapse. However, the three losses can be evolved simultaneously, using performance and reliability as fitness. In one such experiment on generating building facade images given the overall design as a condition, the TaylorGLO approach was found to result in better structural similarity and perceptual distance than the Wasserstein loss (Gonzalez et al., 2023). Although this result

is preliminary, it suggests that evolutionary loss-function optimization may make more complex learning systems possible in the future.

4.2.2 Activation functions

Early on in the 1980s and 1990s, sigmoids (and tanh) were used almost exclusively as activation functions for neural networks. They had the intuitively the right behavior as neural models, limiting activation between the minimum and maximum values, a simple derivative that made backpropagation convenient, and a theorem suggesting that universal computing could be based on such networks (Cybenko, 1989; Hornik et al., 1989). There were indications, however, that other activation functions might work better in many cases. Gaussians achieved universal computing with one less layer, and were found powerful in radial basis function networks (Park and Sandberg, 1991). Ridge activations were also found to provide similar capabilities (Light, 1992).

However, with the advent of deep learning, an important discovery was made: Activation function actually made a big difference wrt. vanishing gradients. In particular, rectified linear units (ReLUs), turned out important in scaling up deep learning networks (Nair and Hinton, 2010). The linearly increasing region does not saturate activation or gradients, resulting in less signal loss. Moreover, it turned out that in many cases ReLU could be improved by adding a small differentiable dip at the boundary between the two regions, in a function called Swish (Ramachandran et al., 2017). This result suggested that there may be an opportunity to optimize activation functions, in general and for specific architectures and tasks.

Like with loss functions, there is a straightforward opportunity in evolving functions through genetic programming (Bingham et al., 2020). Similarly to loss functions, such an approach can be creative, but also results in many functions that make the network unstable. A more practical approach is to limit the search space to e.g. computation graphs of two levels, with a focused set of operators, that are more likely to result in useful functions. This approach was taken e.g. in the Pangaea system (Bingham and Miikkulainen, 2022). Given a list of 27 unary and seven binary operators, two basic two-level computation graph structures, and four mutation operators, evolution can search a space of over ten trillion activation functions.

However, finding an effective function is only part of the challenge. The function also needs to be parameterized so that it performs as well as possible. While coefficients multiplying each operator can be evolved together with the structure, it turns out that such fine tuning can be done more efficiently through gradient descent. In other words, in Pangaea evolution and gradient descent work synergetically: evolution discovers the general structure of the function, and gradient descent finds its optimal instantiation.

The method is powerful in two ways: it finds general functions that perform better than previous functions (such as ReLU, SeLU, Swish, etc.) across architectures (such as All-CNN, Wide ResNet, Resnet, and Preactivation Resnet) and tasks (such as CIFAR-10, CIFAR-100). However, it is most powerful in discovering activation functions that are specialized to architecture and task, apparently taking advantage of the special requirements in each such context.

Furthermore, performance can be further improved by allowing different functions at different parts of the network, and at different times throughout training (Figure 9). The optimal designs change continuously over time and space. Different activation functions are useful early in training when the network learns rapidly and late in training when fine-tuning is needed; similarly, more nonlinear functions are discovered for later layers, possibly reflecting the need to form a regularized embedding early, and make classification decisions later.

The Pangaea results suggest an intriguing duality: While neural network learning is mostly based on adapting a large number of parameters (i.e. weights), perhaps a similar effect might be achieved by adapting the activation functions over space and time? Perhaps the two mechanisms could be used synergetically? Evolution of the activation function structure provides the foundation for this approach, which still needs to

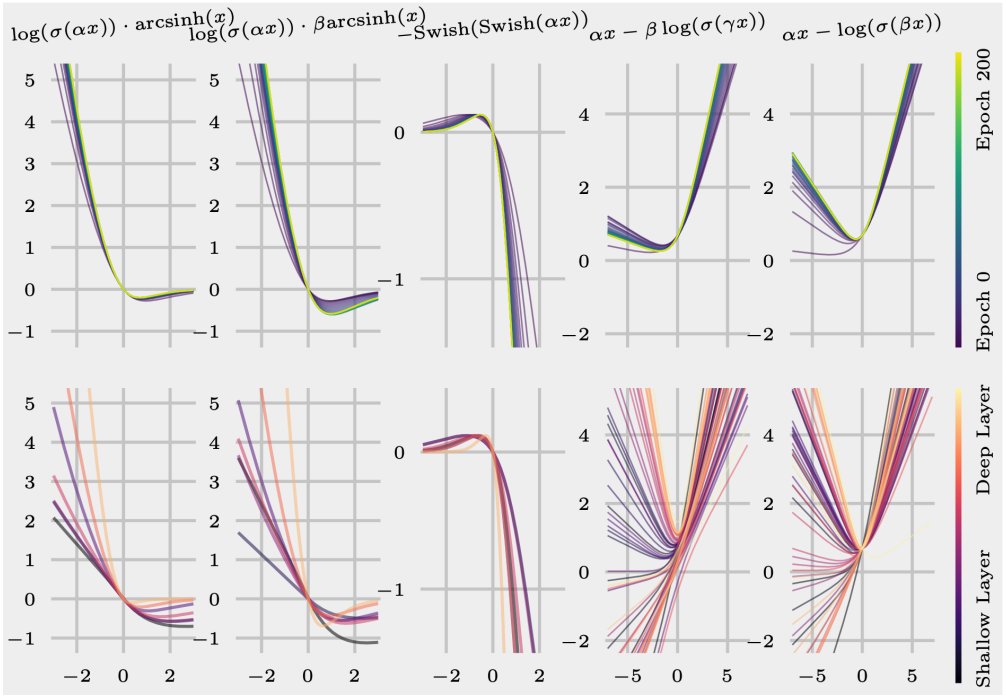


Figure 9: **Activation Functions Discovered over Space and Time.** Pangaea (Bingham and Miikkulainen, 2022) combines evolution of function structure synergetically with gradient descent of its parameters. It is possible to discover general functions, but the approach is most powerful in customizing them to a particular architecture and task. Moreover, the functions change systematically over learning time as well as through different depths of layers, presumably starting with coarse learning and regularization and transforming into fine-tuning and classification. These results suggest a possible duality with weight learning, and a possible synergy for the future. (Figure from Bingham and Miikkulainen, 2022)

be developed fully.

4.2.3 Data use and augmentation

Another important opportunity for evolutionary optimization of supervised learning systems is to optimize the training data. For instance, it may be possible to form embeddings of the training samples through an autoencoder, and then form a strategy for utilizing different kinds of samples optimally through time (Gonzalez et al., 2019). In this manner, evolution could discover ways for balancing an imbalanced dataset or designing curricular learning from simple to more complex examples. Especially in domains where not a lot of labeled samples are available, such techniques could result in significant improvements. It may also be possible to extend the methods to utilize multiple datasets optimally over time in a multitask setting.

Another possibility is to evolve methods for augmenting the available data automatically through various transformations. Different datasets may benefit from different transformations, and it is not always obvious ahead of time how they should be designed. For instance, in an application to develop models for estimating the age of a person from an image of their face, evolution was used to decide vertical and horizontal shift and cutout, as well as a direction of flip operations, angle of rotation, degree of zoom, and extent of shear (Miikkulainen et al., 2021). Unexpectedly, it chose to do vertical flips only—which made little sense for faces, until it was found that the input images had been rotated 90 degrees! It also discovered a combination

of shift operations that allowed it to obfuscate the forehead and chin, which would otherwise be easy areas for the model to overfit.

A particularly interesting use for evolved data augmentation is to optimize not only the accuracy of the resulting models but also to mitigate bias and fairness issues with the data. As long as these dimensions can be measured (Sharma et al., 2020), they can be made part of the fitness, or separate objectives in a multiobjective setting. Operations then need to be designed to increase variance across variables that might otherwise lead to bias through overfitting—for instance gender, ethnicity, and socioeconomic status, depending on the application. While evolutionary data augmentation is still new, this area seems like a differentiated and compelling opportunity for it.

4.2.4 Learning methods

An interesting extension of NAS is to evolve the learning system not from high-level elements, but from the basic algorithmic building blocks (mathematical operations, data management, and ways to combine them)—in other words, by evolving code for supervised machine learning. In this manner, evolution can be more creative in discovering good methods, with fewer biases from the human experimenters.

The AutoML-Zero system (Real et al., 2020) is a step towards this goal. Given an address space for scalars, vectors, and matrices of floats, it evolves setup, predict, and learn methods composed of over 50 basic mathematical operations. Evolution is implemented as a linear GP, and consists of inserting and removing instructions and randomizing instructions and addresses. Evaluation consists of computing predictions over unseen examples.

Starting from empty programs, AutoML-Zero first discovered linear models, followed by gradient descent, and eventually several extensions known in the literature, such as noisy inputs, gradient normalization, and multiplicative interactions (Figure 10). When given small datasets, it discovers regularization methods similar to dropout; when given few training steps, it discovers learning-rate decay.

Thus, the preliminary experiments with AutoML-Zero suggest that evolutionary search can be a powerful tool in discovering entire learning algorithms. As in many metalearning approaches, the main power may be in customizing these methods to particular domains and constraints. A crucial aspect will be to guide the evolution within the enormous search space toward meaningful solutions, without hampering its ability to create, again a challenge shared with most of metalearning.

4.2.5 Synergies

Perhaps the most important future direction in evolutionary metalearning is to discover and utilize synergies between the different aspects of the learning system design. For instance, the best performance was reached by optimization activation functions for the specific architecture; it might be possible to optimize the architecture simultaneously to emphasize this effect.

Simply running evolution on all these design aspects simultaneously is unlikely to work; the search space would be prohibitively large. Similarly, adding more outer loops to the existing process (where supervised learning is the inner loop and metalearning is the outer loop) is likely prohibitive as well. However, it might be possible to alternate evolution of different aspects. Better yet, techniques from bilevel (or multilevel) optimization could be useful—the idea is to avoid full inner-outer loop structure, but instead use e.g. surrogate models to evaluate outer loop innovations (Liang and Miikkulainen, 2015; Sinha et al., 2014).

A practical approach is to simply add constraints, and search in a smaller space. A first such step was already taken in the EPBT system (Liang et al., 2021), which combines hyperparameter tuning, loss-function

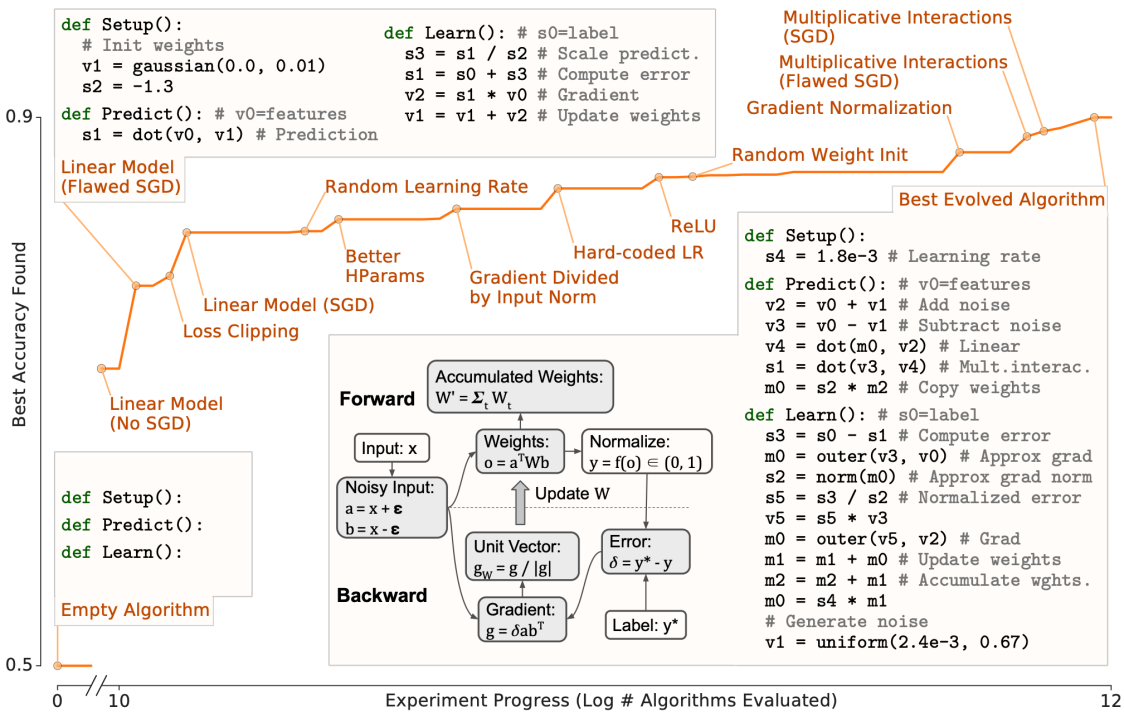


Figure 10: **Evolutionary Discovery of Learning Methods.** In AutoML-Zero (Real et al., 2020), sequences of instructions for setup, prediction, and learning are evolved through mutation-based regularized search. AutoML-Zero first discovered simple methods such as linear models, then several known extensions such as ReLU and gradient normalization, and eventually more sophisticated techniques such as multiplicative interactions. The approach could potentially be useful in particular in customizing learning methods to different domains and constraints. (Figure from Real et al., 2020)

optimization, and population-based training (PBT) into a single loop. That is, hyperparameters and loss functions are evolved at the same time as the networks are being trained. Hyperparameter tuning is limited to those that do not change the structure of the networks (e.g. learning rate schedules) so that they can be continuously trained, even when the hyperparameters change. Similarly, loss-function optimization is limited to TaylorGLO coefficients (Liang et al., 2021) that can be changed while training is going on. Even so, the simultaneous evolution and learning was deceptive, and needed to be augmented with two mechanisms: quality-diversity heuristic for managing the population and knowledge distillation to prevent overfitting. The resulting method worked well on optimizing ResNet and WideResnet architectures in CIFAR-10 and SVHN, but also illustrates the challenges in taking advantage of synergies of metalearning methods.

Similarly, promising results were obtained in an experiment that compared human design with evolutionary metalearning (Miikkulainen et al., 2021). Using the same datasets and initial model architectures, similar computational resources, and similar development time, a team of data scientists and an evolutionary metalearning approach developed models for age estimation in facial images (Figure 11). The evolutionary metalearning approach, LEAF-ENN, included optimization of loss functions (limited to linear combinations of MSE and CE), learning hyperparameters, architecture hyperparameters, and data augmentation methods. Evolution discovered several useful principles that the data scientists were not aware of: focusing data augmentation to regions that mattered most, and utilizing flips only horizontally across the face; utilizing different loss functions at different times during learning; relying mostly on the output level blocks of the base models. With both datasets, the eventual accuracy of the metalearned models was significantly better than

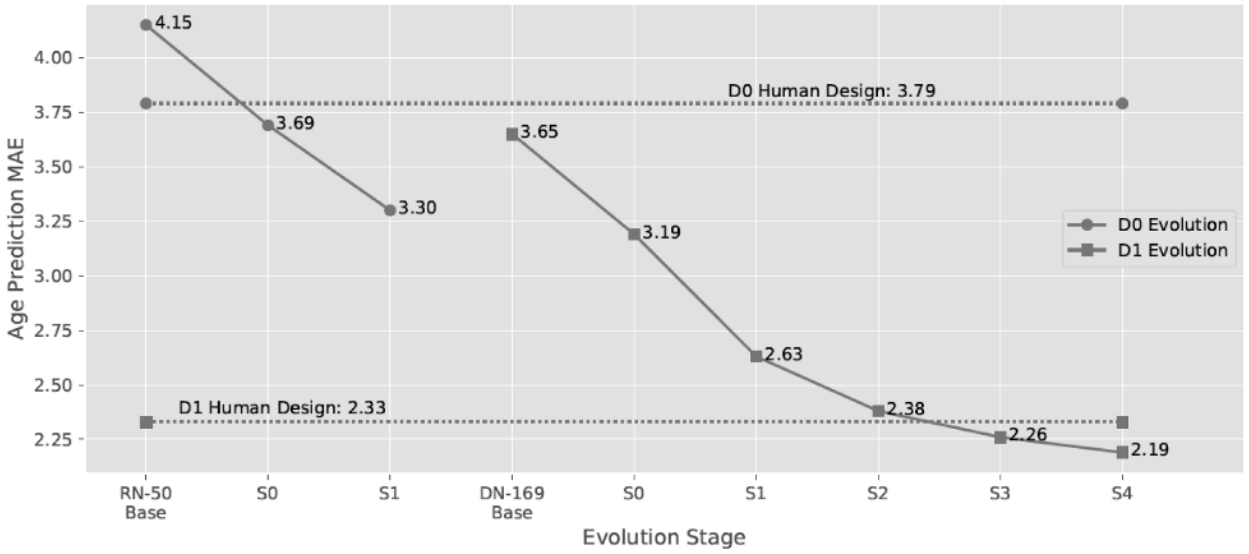


Figure 11: **Utilizing Metalearning Synergies to Beat Human Designers.** In two datasets (D0 and D1) for age estimation from facial images, LEAF-ENN evolutionary metalearning (Miikkulainen et al., 2021) was able to discover models that performed better than those simultaneously designed by human experts. The humans optimized the ResNet-50 architecture for D0 and EfficientNet-B8 for D1. The evolutionary runs progressed in stages: In D0, ResNet-50 (S0) was expanded to Densenet 169 (S1); in D1, DenseNet-169 (S0) was expanded to DenseNet-201 (S1) and trained longer (S2), then expanded to EfficientNet-B6 (S3), and ensembling (S4). At the same time, evolution optimized learning and architecture hyperparameters, data-augmentation methods, and combinations of loss functions. The approach discovers and utilizes synergies between design aspects that are difficult for humans to utilize. The final accuracy, MSE of 2.19 years, is better than typical human accuracy in age estimation (3-4 years). (Figure from Miikkulainen et al., 2021)

that of the models developed by the data scientists. This result demonstrates the main value of evolutionary metalearning: it can result in models that are optimized beyond human ability to do so.

5 Conclusion

Although much of evolutionary machine learning has focused on discovering optimal designs and behavior, it is a powerful approach to supervised learning as well. While gradient-based supervised learning (i.e. deep learning) has benefited from massive scale-up, three opportunities for evolutionary supervised learning have emerged as well. In particular, in domains where such a scale-up is not possible, it can be useful in two ways. First, it can expand the scope of supervised learning to a more general set of design goals other than simply accuracy. Second, it can be applied to solution structures that are explainable. Third, in domains where deep learning is applicable, it can be used to optimize the design of the most complex such architectures, thus improving upon the state of the art. Two most interesting research directions emerge: how to extend the generality and explainability to larger domains, and how to take advantage of synergies between multiple aspects of machine learning system design. Such work most likely requires developing a better understanding of how the search can be guided to desired directions without limiting the creativity of the approach.

References

Aharonov-Barki, R., Beker, T., and Ruppin, E. (2001). Emergence of memory-driven command neurons in evolved artificial agents. *Neural Computation*, 13:691–716.

Aitkenhead, M. J. (2008). A co-evolving decision tree classification method. *Expert Systems with Applications*, 34:19–25.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In Precup, D., and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, 214–223.

Barros, R. C., Basgalupp, M. P., de Carvalho, A. C. P. L. F., and Freitas, A. A. (2012). A survey of evolutionary algorithms for decision-tree induction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42:291–312.

Bi, Y., Xue, B., and Zhang, M. (2022). Genetic programming-based evolutionary deep learning for data-efficient image classification. *IEEE Transactions on Evolutionary Computation*.

Bingham, G., Macke, W., and Miikkulainen, R. (2020). Evolutionary optimization of deep learning activation functions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 289–296.

Bingham, G., and Miikkulainen, R. (2022). Discovering parametric activation functions. *Neural Networks*, 148:48–65.

Breiman, L. (2001). Random forests. *Machine Learning*, 45:5–32.

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. (1984). *Classification and Regression Trees*. Chapman and Hall/CRC.

Butz, M. V., Lanzi, P. L., and Wilson, S. W. (2008). Function approximation with xcs: Hyperellipsoidal conditions, recursive least squares, and compaction. *IEEE Transactions on Evolutionary Computation*, 12:355–376.

Canatar, A., Bordelon, B., and Pehlevan, C. (2021). Spectral bias and task-model alignment explain generalization in kernel regression and infinitely wide neural networks. *Nature Communications*, 12:1914.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314.

Dai, E., Zhao, T., Zhu, H., Xu, J., Guo, Z., Liu, H., Tang, J., and Wang, S. (2020). A comprehensive survey on trustworthy graph neural networks: Privacy, robustness, fairness, and explainability. *arXiv:2104.05605*.

De Jong, K. (1988). Learning with genetic algorithms: An overview. *Machine Learning*, 3:121–138.

Deb, K., and Myburgh, C. (2017). A population-based fast algorithm for a billion-dimensional resource allocation problem with integer variables. *European Journal of Operational Research*, 261:460–474.

Dolotov, E., and Zolotykh, N. Y. (2020). Evolutionary algorithms for constructing an ensemble of decision trees. *arXiv:2002.00721*.

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, 20:1–21.

Gaier, A., and Ha, D. (2019). Weight agnostic neural networks. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, 5364–5378.

Ganon, Z., Keinan, A., and Ruppin, E. (2003). Evolutionary network minimization: Adaptive implicit pruning of successful agents. In Banzhaf, W., Ziegler, J., Christaller, T., Dittrich, P., and Kim, J. T., editors, *Advances in Artificial Life*, 319–327. Berlin, Heidelberg: Springer Berlin Heidelberg.

Gao, B., Gouk, H., and Hospedales, T. M. (2021). Searching for robustness: Loss learning for noisy classification tasks. *IEEE/CVF International Conference on Computer Vision*, 6650–6659.

Gomez, F., and Miikkulainen, R. (1997). Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342.

Gomez, F., Schmidhuber, J., and Miikkulainen, R. (2008). Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, 937–965.

Gonzalez, S., Kant, M., and Miikkulainen, R. (2023). Evolving gan formulations for higher quality image synthesis. In Kozma, R., Alippi, C., Choe, Y., and Morabito, F. C., editors, *Artificial Intelligence in the Age of Neural Networks and Brain Computing (second edition)*. New York: Elsevier.

Gonzalez, S., Landgraf, J., and Miikkulainen, R. (2019). Faster training by selecting samples using embeddings. In *Proceedings of the 2019 International Joint Conference on Neural Networks*, 1–7.

Gonzalez, S., and Miikkulainen, R. (2020). Improved training speed, accuracy, and data utilization through loss function optimization. In *Proceedings of the 2020 IEEE Congress on Evolutionary Computation (CEC)*, 1–8.

Gonzalez, S., and Miikkulainen, R. (2021a). Effective regularization through loss-function metalearning. *arXiv:2010.00788*.

Gonzalez, S., and Miikkulainen, R. (2021b). Optimizing loss functions through multivariate Taylor polynomial parameterization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 305–313.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, 2672–2680.

Hansen, N., and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9:159–195.

Hanson, S. J., and Pratt, L. Y. (1988). Comparing biases for minimal network construction with back-propagation. In *Proceedings of the 1st International Conference on Neural Information Processing Systems*, 177–185. Cambridge, MA, USA: MIT Press.

Hayes-Roth, F. (1985). Rule-based systems. *Communications of the ACM*, 28:921–932.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hemberg, E., Veeramachaneni, K., Wanigarekara, P., Shahrzad, H., Hodjat, B., and O'Reilly, U.-M. (2014). Learning decision lists with lagged physiological time series. In *Workshop on Data Mining for Medicine and Healthcare, 14th SIAM International Conference on Data Mining*, 82–87.

Holland, J. (1986). Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine learning: An artificial intelligence approach*, vol. 2, 593–623. Los Altos, CA: Morgan Kaufmann.

Hoos, H. (2012). Programming by optimization. *Communications of the ACM*, 55:70–80.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366.

Huang, X., Kroening, D., Ruan, W., Sharp, J., Sun, Y., Thamo, E., Wu, M., and Yi, X. (2020). A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review*, 37:100270.

Ijspeert, A. J. (2008). Central pattern generators for locomotion control in animals and robots: A review. *Neural Networks*, 21:642–653.

Jankowski, D., and Jackowski, K. (2014). Evolutionary algorithm for decision tree induction. In Saeed, K., and Snášel, V., editors, *Computer Information Systems and Industrial Management*, 23–32. Berlin, Heidelberg: Springer Berlin Heidelberg.

Kashtan, N., and Alon, U. (2005). Spontaneous evolution of modularity and network motifs. *Proceedings of the National Academy of Sciences*, 102:13773–13778.

Langdon, W. B., Poli, R., McPhee, N. F., and Koza, J. R. (2008). Genetic programming: An introduction and tutorial, with a survey of techniques and applications. In Fulcher, J., and Jain, L. C., editors, *Computational Intelligence: A Compendium*, 927–1028. Berlin, Heidelberg: Springer.

Liang, J., Gonzalez, S., Shahrzad, H., and Miikkulainen, R. (2021). Regularized evolutionary population-based training. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 323–331.

Liang, J., Meyerson, E., Hodjat, B., Fink, D., Mutch, K., and Miikkulainen, R. (2019). Evolutionary neural AutoML for deep learning. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2019)*, 401–409.

Liang, J., Meyerson, E., and Miikkulainen, R. (2018). Evolutionary architecture search for deep multitask networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 466–473.

Liang, J. Z., and Miikkulainen, R. (2015). Evolutionary bilevel optimization for complex control tasks. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2015)*, 871–878.

Light, W. (1992). Ridge functions, sigmoidal functions and neural networks. In *Approximation Theory VII*, 158–201. Boston: Academic Press.

Liu, Y., Sun, Y., Xue, B., Zhang, M., Yen, G. G., and Tan, K. C. (2021a). A survey on evolutionary neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems*, 1–21.

Liu, Z., Zhang, X., Wang, S., Ma, S., and Gao, W. (2021b). Evolutionary quantization of neural networks with mixed-precision. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2785–2789.

Lu, Z., Deb, K., Goodman, E., Banzhaf, W., and Boddeti, V. N. (2020). Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search. In *European Conference on Computer Vision ECCV-2020, LNCS*, vol. 12346, 35–51.

Mao, X., Li, Q., Xie, H., Lau, R. Y., Wang, Z., and Paul Smolley, S. (2017). Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2794–2802.

Meyerson, E., and Miikkulainen, R. (2018). Pseudo-task augmentation: From deep multitask learning to intratask sharing—and back. In *Proceedings of the 35th International Conference on Machine Learning*, 739–748.

Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., and Hodjat, B. (2023). Evolving deep neural networks. In Morabito, C. F., Alippi, C., Choe, Y., and Kozma, R., editors, *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, 293–312. New York: Elsevier. Second edition.

Miikkulainen, R., Meyerson, E., Qiu, X., Sinha, U., Kumar, R., Hofmann, K., Yan, Y. M., Ye, M., Yang, J., Caiazza, D., and Brown, S. M. (2021). Evaluating medical aesthetics treatments through evolved age-estimation models. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 1009–1017.

Montana, D. J., and Davis, L. (1989). Training feedforward neural networks using genetic algorithms. In *International Joint Conference on Artificial Intelligene*, 762–767.

Moriarty, D. E., and Miikkulainen, R. (1997). Forming neural networks through efficient and adaptive co-evolution. *Evolutionary Computation*, 5:373–399.

Nair, V., and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, 807–814.

Oymak, S. (2018). Learning compact neural networks with regularization. In *International Conference on Machine Learning*, 3963–3972.

Papavasileiou, E., Cornelis, J., and Jansen, B. (2021). A systematic literature review of the successors of "neuroevolution of augmenting topologies". *Evolutionary Computation*, 29:1–73.

Park, J., and Sandberg, I. W. (1991). Universal approximation using radial-basis-function networks. *Neural Computation*, 3:246–257.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1:81–106.

Quinlan, J. R. (1993). *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Searching for activation functions. *arXiv:1710.05941*.

Rawal, A., and Miikkulainen, R. (2020). Discovering gated recurrent neural network architectures. In Iba, H., and Noman, N., editors, *Deep Neural Evolution – Deep Learning with Evolutionary Computation*, 233–251. Springer.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2019). Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 4780–4789.

Real, E., Liang, C., So, D., and Le, Q. (2020). AutoML-Zero: Evolving machine learning algorithms from scratch. In III, H. D., and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, 8007–8019.

Reed, R. (1993). Pruning algorithms-a survey. *IEEE Transactions on Neural Networks*, 4:740–747.

Routley, N. (2017). Visualizing the trillion-fold increase in computing power. Retrieved 11/17/2022.

Schaul, T., and Schmidhuber, J. (2010). Metalearning. *Scholarpedia*, 5:4650.

Schmidhuber, J. (2022). Annotated history of modern ai and deep learning. *arXiv:22212.11279*.

Shahrzad, H., Hodjat, B., Dolle, C., Denissov, A., Lau, S., Goodhew, D., Dyer, J., and Miikkulainen, R. (2020). Enhanced optimization with composite objectives and novelty pulsation. In Banzhaf, W., Goodman, E., Sheneman, L., Trujillo, L., and Worzel, B., editors, *Genetic Programming Theory and Practice XVII*, 275–293. New York: Springer.

Shahrzad, H., Hodjat, B., and Miikkulainen, R. (2022). EVOTER: Evolution of transparent explainable rule-sets. *arXiv:2204.10438*.

Sharma, S., Henderson, J., and Ghosh, J. (2020). CERTIFAI: A common framework to provide explanations and analyse the fairness and robustness of black-box models. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, 166–172. New York, NY, USA: Association for Computing Machinery.

Shayani, H., Bentley, P., and Tyrrell, A. (2008). An fpga-based model suitable for evolution and development of spiking neural networks. In *Proceedings of the European Symposium on Artificial Neural Networks*, 197–202.

Sinha, A., Malo, P., Xu, P., and Deb, K. (2014). A bilevel optimization approach to automated parameter tuning. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2014)*, 847–854. Vancouver, BC, Canada.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37, 2256–2265.

Srinivasan, S., and Ramakrishnan, S. (2011). Evolutionary multi objective optimization for rule mining: Areview. *Artificial Intelligence Review*, 36:205–248.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Stanley, K. O. (2004). *Efficient Evolution of Neural Networks Through Complexification*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin.

Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A Hypercube-Based encoding for evolving Large-Scale neural networks. *Artificial Life*, 15:185–212.

Stanley, K. O., and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10:99–127.

Stanley, K. O., and Miikkulainen, R. (2004). Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21:63–100.

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

Suganuma, M., Kobayashi, M., Shirakawa, S., and Nagao, T. (2020). Evolution of Deep Convolutional Neural Networks Using Cartesian Genetic Programming. *Evolutionary Computation*, 28:141–163.

Sun, Y., Xue, B., Zhang, M., and Yen, G. G. (2020a). Evolving deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation*, 24:394–407.

Sun, Y., Xue, B., Zhang, M., Yen, G. G., and Lv, J. (2020b). Automatically designing cnn architectures using the genetic algorithm for image classification. *IEEE Transactions on Cybernetics*, 50:3840–3854.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1–9.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2818–2826.

Urbanowicz, R., and Moore, J. (2009). Learning classifier systems: A complete introduction, review, and roadmap. *Journal of Artificial Evolution and Applications*, 2009:736398.

Urbanowicz, R. J., Bertasius, G., and Moore, J. H. (2014). An extended michigan-style learning classifier system for flexible supervised learning, classification, and data mining. In *International Conference on Parallel Problem Solving from Nature*, 211–221. Springer.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, vol. 30, 6000–6010.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3156–3164.

Wu, X., Zhang, X., Jia, L., Chen, L., Liang, Y., Zhou, Y., and Wu, C. (2021). Neural architecture search based on cartesian genetic programming coding method. *arXiv:2103.07173*.