M. Sidorko, Teacher
N. Kolisnichenko, Teacher
I. Kondratiuk, Bachelor Student
Berdychiv Applied College of Industry, Economics and Law

EVOLUTIONARY ALGORITHMS IN ARTIFICIAL INTELLIGENCE: GENETIC PROGRAMMING AND NEUROEVOLUTION

Evolutionary algorithms are viewed as a set of computational methods inspired by the natural process of evolution [1, c. 3]. Evolutionary algorithms techniques work by improving a population of candidate solutions over multiple generations. Unlike traditional optimization methods, evolutionary algorithms do not require gradient information or smooth, continuous functions; they are effective in handling complex, non-linear, or even discrete problems. At the heart of these methods is a fitness function that evaluates how well each candidate solution performs on a specific task. In genetic programming (GP), computer programs are represented as tree structures — each node representing an operation or a variable. The fitness function scores each program based on its ability to solve the problem, such as predicting a data trend or controlling NPC. The best- performing programs are selected to form a new generation, mimicking the process of natural selection.

The creation of new candidate programs relies on two key operators: crossover and mutation [2, c. 30]. Crossover takes two parent programs and combines parts of their tree structures to produce offspring that inherit traits from both parents. Mutation introduces small random changes into a program, such as altering an operation or replacing a sub-tree with a newly generated one. These operations help maintain a diverse population and allow the exploration of various regions in the solution space. Over time, the evolutionary process can yield innovative programs that outperform those designed manually. Neuroevolution applies these evolutionary ideas to artificial neural networks. Traditional neural network training involves adjusting fixed architectures using methods like gradient descent. In contrast, neuroevolution allows both the weights and the structure of the network to evolve. A well-known method in this area is NEAT (NeuroEvolution of Augmenting Topologies). NEAT starts with simple networks and gradually complexifies them by adding nodes and connections as needed. This approach not only tunes the connection strengths but also adapts the network architecture to the problem, potentially leading to more efficient and tailored solutions.

One clear advantage of neuroevolution is its ability to discover network architectures automatically. Instead of manually designing a network, which can be time-consuming and error-prone, neuroevolution explores many configurations and identifies those that perform best on the task. This adaptability is especially useful in environments where the ideal network structure is unknown or may change over time.

Example: Applying Evolutionary Algorithms to YouTube Recommendation Systems [3]

To illustrate the practical potential of evolutionary algorithms, consider the case of YouTube’s recommendation system. Although YouTube primarily uses deep learning techniques and collaborative filtering, the underlying idea of automatically optimizing a complex system is very similar to evolutionary approaches. YouTube’s recommendation system works in two main stages: candidate generation and ranking. In the candidate generation phase, a large number of videos are selected based on the user’s history,

preferences, and behavior. Then, in the ranking phase, these videos are ordered based on predicted user engagement metrics like click-through rate, watch time, and likes. Imagine applying neuroevolution to optimize the candidate generation stage. In this scenario, the architecture of the neural network responsible for generating candidate videos is not fixed. Instead, an evolutionary algorithm could be used to evolve different network configurations. The fitness function in this context might combine several performance metrics: for example, higher click-through rates and longer watch times would result in a higher fitness score. Over successive generations, the algorithm would produce network architectures that are better at predicting which videos the user is likely to enjoy.

Similarly, genetic programming could be used to evolve decision-making rules within the recommendation pipeline. For instance, a set of rules determining which video features (such as thumbnail quality, video length, or user comments) are most predictive of engagement could be evolved over time. The system would evaluate different rule sets and select those that lead to better overall performance in terms of user satisfaction. By incorporating evolutionary algorithms, the recommendation system could automatically discover new, more effective strategies for selecting and ranking videos. This approach would reduce the need for manual tuning and could help the system adapt more rapidly to changes in user behavior or content trends. Although YouTube's actual recommendation system is proprietary and relies on well-established deep learning models, the example shows how evolutionary algorithms can be applied in similar large-scale, real-world scenarios.

However, there are also some challenges when speaking about Neuroevolution. Evolutionary algorithms often require significant computational resources because they involve evaluating many candidate solutions over numerous generations. This can be a major drawback when each evaluation is computationally intensive, as is often the case with large-scale systems like YouTube's recommendation engine. Furthermore, there is a risk of premature convergence, where the population becomes too uniform and gets stuck in suboptimal solutions.

In summary, evolutionary algorithms like genetic programming and neuroevolution offer powerful tools for developing adaptive, intelligent systems. Their ability to automatically evolve solutions — whether they are computer programs or neural network architectures — makes them a valuable area of research in artificial intelligence. The example of applying these methods to a YouTube-style recommendation system highlights their potential in real-world applications. While challenges such as high computational costs and the risk of premature convergence remain, ongoing research and hybrid strategies continue to expand the practical utility of evolutionary algorithms. As these methods mature, they are likely to play an increasingly important role in optimizing complex systems and improving decision- making processes across various fields [2, 135].

REFERNCES

1. Koza, J.R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press.

2. Mitchell, M. (1996). An Introduction to Genetic Algorithms. MIT Press.

3. CGP Grey. (2017, December 18). How Machines Learn [Video]. YouTube. https://www.youtube.com/watch?v=R9OHn5ZF4Uo&ab_channel=CGPGrey