Iterative Data-driven Optimizing Behavior (IDOB): A Structural Model for Enhancing Machine Learning Education and Understanding Software 2.0 Production

Iterative Data-driven Optimizing Behavior (IDOB)

Uwe Lorenz

Freie Universität Berlin, Königin-Luise-Str. 24-26, 14195 Berlin, uwe.lorenz@fu-berlin.de

Ralf Romeike

Freie Universität Berlin, Königin-Luise-Str. 24-26, 14195 Berlin, ralf.romeike@fu-berlin.de

In this contribution, we introduce the Iterative Data-driven Optimizing Behavior (IDOB) scheme, a structural model for didactical purposes, that provides a more comprehensive understanding of Machine Learning (ML) and helps to explain how "Software 2.0", software that is a product of data-driven ML techniques, is generated. Our objective is to present a didactical scheme that students help more easily construct new information about any ML methods, such as details of specific algorithms, according to the idea of active knowledge construction upon appropriate pre-conceptions by the learner. We justify our proposal theoretically and by highlighting its applicability across major ML paradigms (Reinforcement, Unsupervised and Supervised Learning), show its ease of use by examples using the Snap! programming language and its potential in interdisciplinary, hybrid software systems, where it facilitates comparative reflections on ML methods and Software 2.0 versus alternative solutions, such as manually implemented models or mathematical representations.

Keywords and Phrases: AI, artificial intelligence, machine learning, Software 2.0, computer science education, Snap!

1 INTRODUCTION

Writing programs that emulate human abilities can be one of the most motivating aspects of working with "Artificial Intelligence" (AI). Perhaps it is comparable to the task of developing the abilities of a fantasy character. However, and especially due to the recent very successful subfield of Machine Learning (ML), another aspect of AI has come to the foreground that is much more important from a Computer Science perspective, where a central aim is to make computers more useful: the automatic creation of software functions, or automatic problem-solving through interaction with a data source or dataset (Jormanainen et al., 2023; Karpathy, 2017; Schulte et al., 2018; Tedre et al., 2021). It has been shown that with these ML methods, it is possible to automate many tasks that could not be done with traditionally produced software, or that for quite a few tasks it is easier to apply ML methods with suitable data sets than to figure out the necessary rules for a traditional program (Jormanainen et al., 2023). Software that is produced in this way is referred to by Karpathy as "Software 2.0" (Karpathy, 2017).

So teaching ML is not only important to improve future skills related to a set of new spectacular applications, such as evaluating the outputs of ML applications, making them more useful and innovating with them, and thinking

about their possible impact on society, their limitations, and possibilities, etc. but we seem to be witnessing the rise of a new paradigm for developing software and making computers useful in general. Classical problem-solving skills apparently no longer have the same significance here (Tedre et al., 2021).

In constructionist educational settings, students should not just work on puzzles or problem-solving activities, but on meaningful projects based on their passions, in collaboration with peers, and with a playful spirit (Resnick & Rusk, 2020). So AI tools and methods that can relieve this type of activity can fit well with the constructionist approach. They offer incredibly rich new opportunities for the creation of computational artifacts, including artwork of personal and social significance that have not been possible before - especially for children.

But there are also fundamental challenges that must be considered in the context of constructionist approaches. Even small children can start using these systems intuitively before they have learned anything about them (Tedre et al., 2021). Due to the automatic production of "Software 2.0", tasks like handling, and preparation of data in the context of a Machine Learning Life Cycle process move to the center. In the case of ready-trained universally or at least very versatile applicable "Software 2.0" like Large Language Models (LLM) and Generative Systems, it is "something of a black art" (Norvig, 2022) to create suitable inquiry prompts. While addressing such issues that arise during Software 2.0 building or its use, these activities can provide little insight into how ML engines work.

Besides the fact that these also would certainly help e.g. to understand the necessary properties of the data to be processed or to develop sustainable prompt engineering competencies, there is a risk that some important elements of constructionist learning will be lost. For Minsky and Papert it was important that the acquisition of concepts by interacting with computers should at the same time foster their own ability to reflect and learn. Kahn and Winters, among others, emphasize this aspect of dealing with "AI" - and here also understood in a broader sense as a simulation of cognitive abilities - namely that modeling AI can lead to reflection and to a better understanding of cognitive processes (Kahn & Winters, 2021) such as perceiving, thinking, decision-making, or learning. It remains important and exciting to learn how it is that ML systems "learn" things. It is desirable for children to ask questions such as "How does this learning in the computer actually work?", "Can I build something like that myself?" or "Does it perhaps say something about how I learn?" There is, even if it cannot be assured, the hope that with questions like these, children can make themselves better thinkers and learners (Papert, 1986). We should not deprive them of the chance to ask such questions. Furthermore, understanding how AI processes content-related data and thus builds or applies its internal modeling could also provide new insights into domain-specific understanding. But is it even possible to gain such insight, given the vast array of complex algorithms and approaches, and the speed at which this technology is evolving?

2 THEORETICAL FRAMEWORK

The answer we give here to this question is yes, but there are some prerequisites for engaging with this field to make sense in general education contexts. To this end, we will address in this paper four questions that lead from an abstract idea of how machines learn to a concrete modifiable interdisciplinary scenario that enables play in an instructive way.

Powerful Idea: In settings designed to give a behind-the-scenes look, it should be avoided that students only deal with single ones of the numerous ML algorithms and methods, which may become obsolete in a short time, and have to deal with their details without discovering central concepts with relevance in the context of general education. Are there "Powerful Ideas" that underlie ML in general, i.e., that are also valid in the long term and all paradigms?

Structural model for comprehensive understanding: A structure that models the basic components and their interactions that are recognizable in all ML systems, possibly in different manifestations or forms, would be conducive to learning and understanding because such a structure could allow students to more easily construct new information about any ML methods according to the idea of active knowledge construction based on pre-concepts. What can such a structural model for all ML look like?

Implementation: We need learning environments that make it clear how the essential components work and also encourage people to engage with different manifestations of them, e.g., to implement innovative or outstanding approaches to solutions and hopefully discover Powerful Ideas. What do examples of concrete implementations look like?

Comparisons to other solutions in hybrid and interdisciplinary application contexts: When a learning system is applied in a particular context, it is faced with data sets or data sources about which the ML system "learns" something and with which the ML system adopts a behavior desired by the developer. What are the properties of the representation generated by the ML system and the corresponding behavior of the system? How can the properties of such automatically produced solutions be compared with other solutions, e.g., of manually programmed software or explicit calculation rules (mathematical formulas)?

ML as a software development paradigm

A look at the currently used AI definitions shows that emulating intelligence or human abilities is often seen as the essential aspect of AI. This idea in principle also underlies the “5 Big Ideas in AI” (Touretzky et al., 2019). In the current “AI-watch” report (European Commission. JRC, 2020) most of the textual listed definitions are based on psychological terms. A definition of “AI” based on the idea of simulating human cognitive abilities entails some fundamental problems from the point of view of Computer Science (CS), so this cannot fulfill the criterion of unambiguity, e.g. 70 different definitions are given for "intelligence" in (Legg & Hutter, 2007), nor of delimitation since CS has been dealing with the simulation of cognitive activities at least since the first mechanical computing machines. The prevailing notions about AI and also many "definitions" of the term evoke false anthropomorphist associations, which produce misunderstanding and hinder education about the topic (Garside, 2023).

In Karpathy (2017), a qualitative distinction is made between "Software 2.0" and conventional software "Software 1.0" based on their development method. Software 1.0 is created using common software development methods, with phases such as specification, design (e.g. top-down or bottom-up approach), implementation, testing, etc. Software 2.0 is a product of data-driven ML techniques, and the development process here looks quite different and should follow the phases of a Machine Learning Life Cycle with phases like gathering and validating data, dividing data into training, development and test sets, choosing model class(es), train models on training data, validate models on development set, evaluate on test data, deploy, etc. (Norvig, 2022). Karpathy describes the generating process for this type of software as a search in the space of possible functions, which is a key difference from “knowledge-based AI” where suitable outputs are searched for. The product in ML approaches is code that can be executed by the ML engine and continuously evaluated and optimized. However, the codes that are data-driven generated by the ML-Systems are often hardly interpretable by humans, since the resulting code of the "Software 2.0", has often the form of innumerable parameters and requires a corresponding ML engine to be executed. Due to hardware-technical but also informatic conceptual developments, as well as the easy availability of appropriate datasets, the corresponding methods have become so good that they substitute or even significantly surpass the capabilities of traditional software development in larger problem classes (Karpathy, 2017).

We follow here a view according to which an alternative problem-solving paradigm is applied in ML systems, in which the information-processing process that solves a problem or produces desired outputs need not be described but is data-driven generated. As far as we are aware, there are no contributions from the field of CSE that use the definition of ML as a kind of software development and problem-solving method based on iterative optimization of some kind of initial software for providing a unified explanation of all machine learning.

3 ITERATIVE DATA-DRIVEN OPTIMIZING BEHAVIOR (IDOB)

3.1 Powerful idea

The idea of iteratively optimizing behavior by reducing errors or maximizing rewards is central to ML methods as a whole (Mitchell, 1997). Recent works on the foundations of ML (Jung, 2022) also state that all ML methods operate in such a way, but that they are based on different design decisions for data, loss, and model. According to Jung, ML methods combine these three components within a principle that consists of the continuous adaptation of a hypothesis about a phenomenon that generates data. The adaptation or improvement of the hypothesis is based on the discrepancy between predictions and observed data. The ML methods use a loss function to quantify this discrepancy. According to the "intelligent agent paradigm of AI" (Russell & Norvig, 2021), any purposeful, goal-oriented behavior can be considered "intelligent" to some degree. In (Silver et al., 2021) it is hypothesized that intelligence in general, and its associated abilities, can be understood as subserving the maximization of reward. According to the intelligent agent paradigm of AI, all ML can be understood as improving goal-directed system behaviors by iteratively maximizing a "goal function" through processing feedback (Lorenz & Romeike, 2022).

3.2 Structural model for a comprehensive understanding

In an agent-oriented view of problem-solving, the learner imagines the computer to be an agent that produces preliminary error-prone outputs based on the available representations and with its input ("perception") and then receives feedback with which the goal function produces a loss/reward that the Optimizer uses to improve the model and hence future behavior. Based on these considerations, it is proposed to use a principle of Iterative Data-driven Optimizing (IDOB) as an underlying principle for presenting ML methods and algorithms.

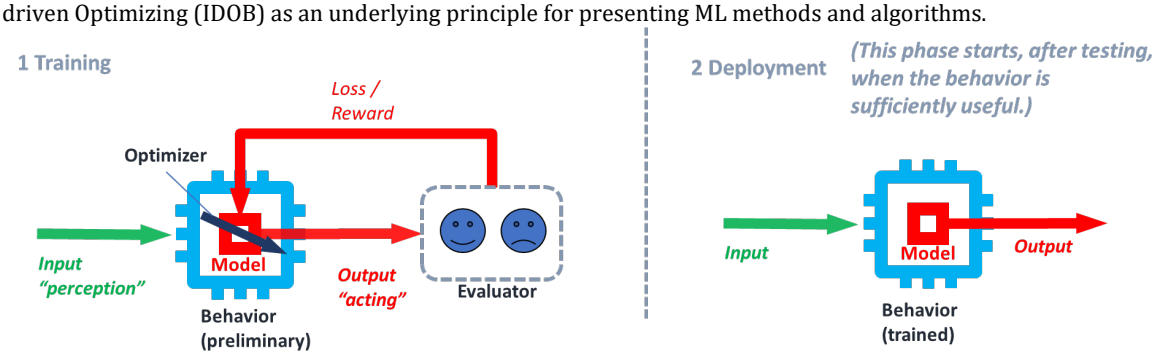

Figure 1 Operating phases of an ML-System: (1) generation and (2) deployment of "Software 2.0". Shown are the operational components of IDOB (Behavior, Evaluator, and Optimizer) and central data flows.

In all IDOB an Optimizer updates the model based on assumptions about how to improve system behavior using the loss generated by the Evaluator's goal function. The different ML paradigms, like Supervised Learning,

Reinforcement Learning, and Unsupervised Learning, can be integrated into the scheme by considering their different types of feedback and corresponding design decisions for Data, Loss, and Model (Jung, 2022). Supervised Learning: The Optimizer minimizes the "Loss" between true labels and those predicted. Reinforcement Learning*:* The Optimizer tries to maximize the cumulative reward of episodes the agent receives from its environment during its interactions. Unsupervised Learning*:* Optimizer here tries to minimize the costs of a model to represent a dataset appropriately, e.g., resources such as memory or computation time, or errors due to improper generalization that diminish predictive ability. Central components of an ML system in the IDOB scheme (cf. Figure 1) are:

Operational components:

Behavior: Produces the output based on the input and internal representations.

Evaluator: contains a goal function and provides the Loss or Reward

Optimizer: uses the Loss/Reward to improve the behavior by updating the model.

Data structures:

Input ("perception"): Sensory input data used to generate the output and, if necessary, to improve future behavior.

Model ("representation"): Contains "software 2.0" encodings that can be interpreted to generate system output.

Output ("acting"): The output that is as appropriate as possible with respect to the given input. Deficits with respect to the desired behavior can be used to generate the Loss.

Loss ("reward"): Represents an evaluation of the preliminary behavior used by the optimizer to improve the system behavior.

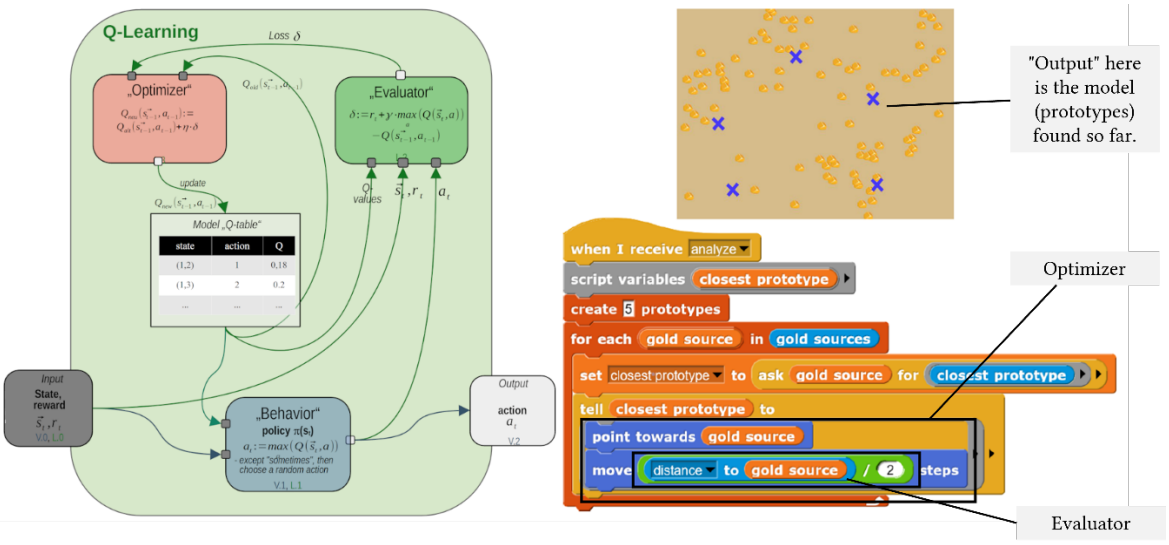

Figure 2 Recognizing IDOB components: Reinforcement Learning (Left: Q-Learning as a data-flow model) Unsupervised Learning (Right: "Gold rush clustering" (Michaeli et al., 2020))

3.3 Implementation using Snap!

Block-based programming embodies the principles of active, hands-on learning while providing a supportive environment for exploration, collaboration, and creativity, focusing on concepts rather than syntax. A predecessor of Snap! *(Snap!)* is called BYOB, an acronym that stands for "Build Your Blocks", indicating that it can be used to

define new blocks yourself, e.g. for the IDOB components. In comparison with Scratch Snap! also offers some advanced features that allow a deeper exploration of computer science concepts.

We present an example inspired by topics from middle school chemistry and physics classes ("flash point" and "free fall") including ML in the IDOB scheme (Figure 3). The idea in the chemistry example was that students teach their AI the concepts of flash point and ignition temperature. The perceptron learns here a logical "AND" function. The feedback in this scenario is provided by a "supervisor", which symbolizes that the correct answer is known during the training with "Supervised Learning". An interesting learning effect can also result from placing the perceptron in a different context, where another, e.g. an OR, relation is to be learned. In this case, not the ML system has to be changed, but the behavior of the "supervisor" must be reprogrammed, which leads to the idea that the same system can code and adapt different functions. A modification task that encourages changes in the setup could be concerned with working out the difference between flash point and ignition temperature by adding new conditions and teaching the robot to behave correctly. The limits may quickly become apparent when transferring the perceptron into further contexts. Multi-layer perceptrons (MLP) can overcome these limitations, but the programming on which the IDOB components are based now requires some mathematical knowledge (sigmoid, derivative), c.f. Figure 4.

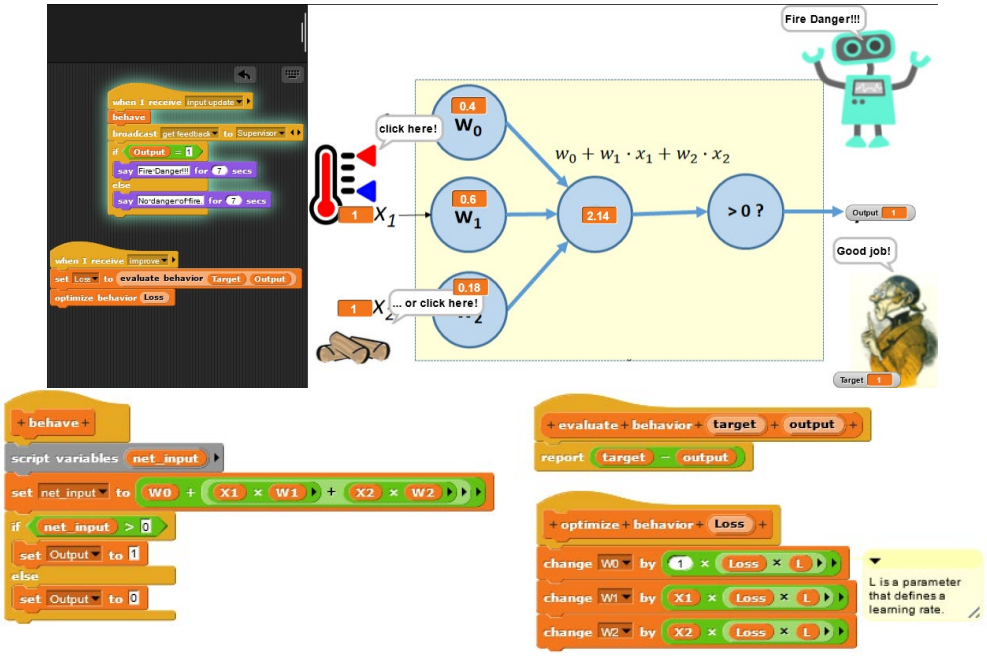

Figure 3 Separate representations of IDOB components: the Behavior component that converts inputs into outputs, the Evaluator that generates the Loss, and the Optimizer that updates the Model, making the processes of "acting" and "learning" transparent and understandable.

3.4 IDOB combined with other approaches in interdisciplinary contexts

The MLP can "learn" a simulated physical process, in this case, "free fall". The scenario is handled by various models, that interact with each other. They all can be inspected, modified, or created: an implementation for the simulation as Snap! program (free fall of a ball as default in this case), mathematical descriptions (formulas, function curve),

and an MLP in IDOB that can approximate arbitrary functions in a f(x) = y form. It is used here for predictions of the "measurement" results. In this scenario, measurement data must first be collected "experimentally" using the free fall simulation for measuring falling times from different heights, which could also be combined with a real-world activity. Finally, the training of the IDOB system can be started, which stops when all data points displayed are touched by the regression curve. A robot is included here that leads through the scenario. It also implements a small competitional setting here where predictions between the robot and the user can be compared. Students can solve the prediction problem better than the AI if they think about the explicit calculation rule because the MLP just does a rough approximation. The robot encourages the students to do so. The curve of the proposed formula can also be added to the coordinate system. The setup provides several opportunities to encourage students to transform different representations into each other. This could include not only mathematical formulas and the related curves but also the simulation program and the approximated ML model. In mathematics didactics, it is a common term for understanding when students can do so. This transparent combination of content and methods from different disciplines allows for an instructive, critical, reflective comparison not only in terms of the nature of the descriptions and predictions, but moreover in reflections in terms of properties such as effort, reliability, or ethical concerns.

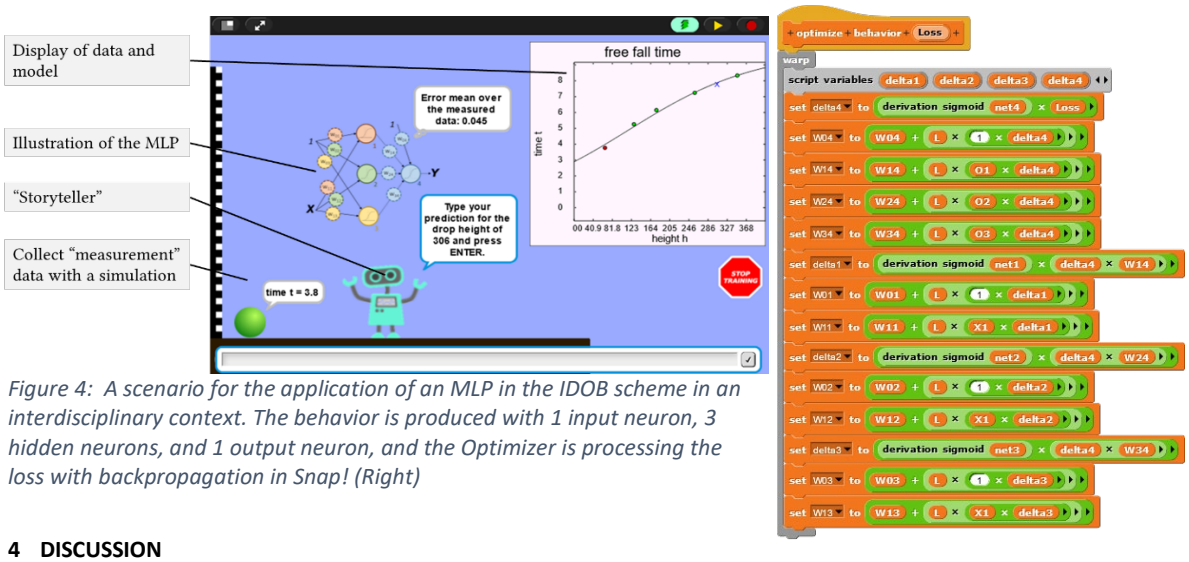

Figure 4: A scenario for the application of an MLP in the IDOB scheme in an interdisciplinary context. The behavior is produced with 1 input neuron, 3 hidden neurons, and 1 output neuron, and the Optimizer is processing the loss with backpropagation in Snap! (Right)

4 DISCUSSION

It was intended to present a scheme that allows a better understanding of ML techniques and the production processes of "Software 2.0". Our model consists of three key basic components, Behavior, Evaluator, and Optimizer, which highlight how ML systems learn and operate. The proposed scheme can be found in the central paradigms of ML such as Reinforcement Learning, Supervised and Unsupervised Learning, is easy to use and it is shown that an application in hybrid software systems in interdisciplinary contexts is also possible, which also allows a comparative reflection of ML with respect to other solutions such as manually implemented software.

We tried the scenario with two grade 12 high school students (1 male and 1 female) and student teachers from a CSE seminar. The students indicated that the Snap! scenarios allowed them to gain a deeper understanding of how neural networks work, particularly backpropagation. An interesting observation regarding supportive design features was that when they attempted to change activation functions or learning rules, they first attempted to interact directly at the Snap! stage with the passive illustration of the neurons. This indicates opportunities for a

more intuitive design based on direct interaction capabilities with the computational nodes and their connections, combined with an immediate view and more intuitive ways to examine system states and processes. Perhaps data-flow modeling (c.f. Fig 2), as seen in many materials related to ML, could illustrate a better mapping of how the components interact than the sequential scripts.

LLMs are "Software 2.0" programs that have been trained to find useful answers to input prompts using a large corpus of text. They can directly perform a variety of tasks and solve problems of various kinds without training because they are trained with knowledge of a large portion of the Internet. Since they are products of ML techniques, they can be thought of as a kind of "Software 2.0" problem solvers or expert systems. They use a combination of techniques e.g. Unsupervised ML to pre-train a model with large text sets, Supervised learning for training an Evaluator component, and Reinforcement Learning to fine-tune the Behavior for more natural interaction. IDOB could help to understand the modular structure of such AI systems, but some development and empirical research would be needed here.

5 REFERENCES

Garside, B. (2023, April 13). *How anthropomorphism hinders AI education*. https://www.raspberrypi.org/blog/ai-education-anthropomorphism/

Jormanainen, I., Tedre, M., Vartiainen, H., Valtonen, T., Toivonen, T., & Kahila, J. (2023). Learning Machine Learning in K-12. In S. Sentance, E. Barendsen, N. R. Howard, & C. Schulte (Hrsg.), *Computer Science Education: Perspectives on Teaching and Learning in School* (2. Aufl.). Bloomsbury Academic.

Jung, A. (2022). *Machine Learning: The Basics*. Springer Nature Singapore.

Kahn, K., & Winters, N. (2021). Constructionism and AI: A history and possible futures. *British Journal of Educational Technology*, *52*(3), 1130–1142.

Karpathy, A. (2017). *Software 2.0*. https://karpathy.medium.com/software-2-0-a64152b37c35

Legg, S., & Hutter, M. (2007). *A Collection of Definitions of Intelligence*.

Lorenz, U., & Romeike, R. (2022). Addressing challenges of constructionist modeling of adaptive systems. *Proceedings of the 17th Workshop in Primary and Secondary Computing Education*, 1–2.

Michaeli, T., Seegerer, S., Jatzlau, S., & Romeike, R. (2020). Looking Beyond Supervised Classification and Image Recognition – Unsupervised Learning with Snap! *Constructionism 2020: Exploring, Testing and Extending our Understanding of Constructionism conference proceedings*.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Norvig, P. (2022, Oktober 5). *Education for AI and by AI*. Stanford HAI Weekly Seminar. https://hai.stanford.edu/events/peter-norvig-education-ai-and-ai

Papert, S. (1986). *On Logo New Mindstorms, tape 1—Resonances*. https://el.media.mit.edu/logo-foundation/resources/onlogo/newmindstorms1.html

Resnick, M., & Rusk, N. (2020). Coding at a crossroads. *Communications of the ACM*, *63*(11), 120–127.

Schulte, C., Sentance, S., & Barendsen, E. (2018). Computer Science, Interaction and the World. In *Computer science education: Perspectives on teaching and learning in school* (S. 57–71). Bloomsbury Academic.

Snap! [Software]. UC Berkeley and SAP. Accessed at 2023, August 4th, https://snap.berkeley.edu/

Tedre, M., Denning, P., & Toivonen, T. (2021). CT 2.0. *21st Koli Calling International Conference on Computing Education Research*, 1–8.

Touretzky, D., Gardner-McCune, C., Martin, F., & Seehorn, D. (2019). Envisioning AI for K-12: What Should Every Child Know about AI? *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*, 9795–9799.