WASP PhD Course on Software Engineering and Cloud Computing - Module 2 Assignment 2

Robert Gieselmann, robgie@kth.se

2022

My PhD research is at the intersection of machine learning and robotics. I am interested in developing new data-driven control methods to improve the current state of the art in robot object manipulation, especially the control of deformable objects such as textiles or cables. The development of such methods is difficult due to high-dimensional state observations, unknown dynamics and complex mechanical phenomena such as friction and elasticity. From an algorithmic standpoint, this often leads me to a reinforcement learning problem formulation. In this subfield, I am particularly curious about the interplay between planning algorithms and learning. Traditional motion planning methods rely on compact state representations, known dynamics, and well-described robot configuration spaces. In one of my recent papers, I combine motion planning algorithms from the robotics literature with concepts from self-supervised contrastive learning to enable planning from high-dimensional video input. In my day-to-day research I work with various machine learning tools such as density estimation, deep generative modeling, representation learning, imitation and reinforcement learning.

Sustainable Software Engineering Sustainable software engineering is an emerging field that addresses the design, implementation, and deployment of energy-efficient software products to reduce climate impact. With respect to machine learning software, sustainability is particularly important because modern deep learning models often consist of billions of parameters that require days of optimization on specialized hardware. A single training run not only involves financial costs, but also raises questions about environmental impact, as a lot of energy is consumed in several phases: for powering the machines during training and execution, for the resources and rare metals needed for the hardware, for manufacturing the hardware, etc. It is therefore important to design sustainable machine learning programs and even consider the sustainability of the development process itself. Reflecting on my own research, I admit that I make extensive use of my university department's GPU cluster and train a large number of models. To obtain the best-fitting model, I perform hyperparameter optimization, which requires training the same model multiple times for different configurations. Advances in available optimization tools and methods (e.g., replace backprop for neural networks) may help reduce computation for training of machine learning models. Nevertheless, the size and thereby computation needed for optimizing state-of-the-art models has increased significantly in recent years and is likely to increase even more. In general, I believe that scientists in my field do not consider or know the environmental impact of their research. As a starting point, we should be made aware of the negative impacts. For example, it would be useful to include tools which estimate the environmental impact for a training run (e.g. carbon footprint) into existing machine learning frameworks.

Automated Software Testing While manual software testing is performed step-by-step by a human, automated software testing is performed with the help of specialized testing software tools. Their main goal is to reduce human input, achieve better test coverage, resource efficiency and accuracy of test results. In my daily work, I spend most of my time writing code for scientific experiments using Python or C++, for example, implementing a new machine learning algorithm, creating a physics simulation or a data collection script. Even though the code I write is for prototyping purpose, testing is still important to verify the correctness of the experimental results and provide clean code which can be used by other researchers. However, I do not use automated software testing tools at the moment since the cost of their implementation overshadows the actual benefits in my case. Note that software testing is generally challenging for machine learning projects because of the enormous uncertainty involved. Compared to traditional programs, machine learning software is often less deterministic, which makes designing useful test more difficult. Another important aspect is the strong dependence on the input data, which significantly increases the extent of testing. An interesting future direction is the use of machine learning tools to bear with the increased complexity of testing machine learning software. For instance, one could leverage powerful learning methods to exploit patterns in the input data to automatically synthesize useful tests.

Security and Privacy Software security is about robustness against external attacks on the system. The general shift toward machine learning systems has created new challenges for developers to ensure robustness against malicious attacks. Machine learning models, such as deep neural networks, are often considered general-purpose tools and black-box models. Once trained on a large dataset, the complexity of the model and the large number of parameters make it difficult for humans to understand the internal reasoning process and clearly understand why a particular outcome was predicted. At the same time, many learning methods suffer from poor generalization beyond the support of the training data. This brittleness and the lack of interpretability cause potential security risks. For example, hackers could manipulate the network's input to target specific predictions that lead to the failure of the entire system. In robotics applications, this could lead to a physical robot making decisions that cause harm to its environment or humans. In the literature, such type of malicious attacks are often called adversarial examples, i.e. input samples which cause a machine learning system to make wrong predictions with high confidence. It is therefore critical to develop interpretable machine learning systems in the future. Moreover, most current deep learning models make predictions with high confidence without measuring uncertainty. An exciting future research direction is how to equip models with the ability to accurately measure their own uncertainty about an input query and detect anomalies. Another concern in the development of machine learning systems is privacy. It refers to the protection of individual's data, such as personal information, e.g., user data. Deep learning models consume large amounts of data during training, and knowledge is implicitly stored in neural network weights. If hackers have access to the network parameters, they might be able to extract specific information about individual users. In generative models, it is important to ensure that the model does not simply remember the data of individual users. Therefore, an interesting research direction is how to develop robust models that cannot be exploited to reveal private information contained in the training data.

ML will "eat Software", i.e. most software development will be the training/development of ML models with only minimal "glue" code to integrate models

Machine learning models have begun to replace traditional software in several application areas [1]. Even in my own field, I see a significant shift from hand-developed robotics pipelines to data-driven methods, for example, for robot control and perception. While most of the machine learning in real-world robots is still being used in research, there are a growing number of startups and technology companies showing interest in this new technology. I believe that machine learning will continue to replace traditional software. Software development practices will not disappear, but rather shift to machine learning workflows, such as pre-processing of data, feature selection, etc. That said, there are application areas that are more receptive to the adoption of machine learning models than others. For example, consider software development for safety-critical systems, such as engine control software for autonomous vehicles. Modern machine learning models such as deep neural networks still lack the ability to formally verify and evaluate the correctness and robustness of the system, meaning machine learning models are often unpredictable. Traditional code written by experienced software engineers is still required to guarantee these properties, which will not change in the near future unless completely new algorithms are discovered.

One topic that I am very interested in is the increased use of machine learning systems to create better programming interfaces and support the software development process. This will also impact the way I write code and manage my time. Most popular IDEs already provide support through code completion, but future environments could go further and generate code for entire functions or larger parts of the program. Software engineers would then have to write less code themselves and instead assemble the correct blocks and check their correctness. Such models could be trained on code available on the Internet (Github repositories, etc.) to make appropriate code predictions and relieve software developers of tedious and repetitive coding tasks. In this context, it is also important to design input interfaces that allow engineers to quickly specify the desired features and properties of the code. The most direct approach for us humans to communicate and formulate ideas is through language. Large-scale language models such as GPT-3 and ChatGPT [2] have gained much attention in generating text. Just a few weeks ago, Deepmind introduced AlphaCode [3], a generative machine learning model designed to generate program code at a competitive level [4]. The potential of these systems is enormous, as they could be integrated into existing IDEs to generate program code even for advanced coding tasks.

Recently, large-language models have also been introduced in robotics research to automatically generate code for high-level robot decision-making [5]. The model generates Python-like programs combining logical structures, parameterized control primitives, etc., while the user specifies objectives through natural language commands. The proposed method shows initial success in automatically synthesizing code for robot control, which, unlike black-box neural networks, allows better interpretability of learned behavior. Technological advances in recent years have been fascinating and have had a direct impact on how we develop code and applications in robotics. While these methods will not completely remove the need for humans in the short term, I see great potential to at least speed up the software development process and even allow robots to be programmed without needing to be a specialized subject matter expert.

[1]https://www.forbes.com/sites/cognitiveworld/2019/08/29/software-ate-the-world-now-ai-is-eating-software/?sh=5b93f0175810

[2]//openai.com/api/

[3]https://alphacode.deepmind.com/

[4]https://www.deepmind.com/blog/competitive-programming-with-alphacode

[5]https://code-as-policies.github.io/