Software engineering essay

Viktor Nilsson

October 2022

1 Introduction

My research lies within the application of particle methods from computational statistics into deep learning settings, an area with limited exploration. The fundamental idea is to delegate the computational responsibility of one neural network to several and train them in tandem. This way, each network can be considered a particle in $\mathbb{R}^n$, where n is the number of parameters of the network. Also the inference is performed jointly, why this largely falls under the field of ensemble methods. The most crucial aspect of leveraging several networks is how they should be trained together, and actually benefit from the many.

One major recent line of progress is the succesful application of these methods into generative adversarial networks (GANs) [1]. Formally, one then has particles (networks) of two types, generators $\{X_i\}_{i=1}^n$ and discriminators $\{Y_i\}_{i=1}^n$. The training dynamics follow a system of stochastic differential equations (SDEs) given by

$$\begin{aligned} dX_t^i &= -\frac{1}{n}\sum_{j=1}^n \nabla_x l(X_t^i, Y_t^j)dt + \sqrt{2\beta^{-1}}dW_t^i, \quad X_0^i = \xi^i \sim \mu_{x,0}, \\ dY_t^i &= -\frac{1}{n}\sum_{j=1}^n \nabla_y l(X_t^j, Y_t^i)dt + \sqrt{2\beta^{-1}}d\bar{W}_t^i, \quad Y_0^i = \bar{\xi}^i \sim \mu_{y,0}, \end{aligned}$$

where $\mu_{x,0}, \mu_{y,0}$ are some initial distributions.

2 Software engineering aspects

2.1 Automated software testing

In software engineering, it is important to continuously affirm that the software meets requirements. Each change to the code might induce unforeseen bugs, perhaps in a completely separate part of the program. In order to detect these as soon as possible, and especially before deployment, there should be a rigorous set of tests that the software should pass. These are themselves often implemented in code as unit tests, i.e. a piece of code that runs part of the program with some predetermined set of inputs and then checks that the corresponding output is correct and perhaps other criteria, like performance.

Using unit tests, the software engineer does not have to run them manually. Instead, they can be launched e.g., every time the software is built. If there is a significant test suite, it can be run less frequent in its completeness. In this way, the programmer can think less about recurring bugs and let them be caught automatically be the test suite.

In machine learning, it is less easy to formulate exact notions of what a model "should" do exactly, but one could at least check that it performs well on some subset of the data. Thus, existing pipelines for unit testing could be implemented for machine learning but likely, it has to use more soft rules since 100 % accuracy is rare. However, there is great opportunity for testing certain important cases where models are extra sensitive. One example of this is the Tesla driving assistant that has confused the moon in the night sky with a yellow traffic light [2].

2.2 Sustainability

Training machine learning models is often associated with a heavy energy consumption. As models have grown ever larger and more complicated, while being trained on greater and greater datasets, the computational effort for training them has grown exponentially. Despite improvements in hardware, optimization algorithms and methodology, energy expenditure has grown into a major societal burden caused by artificial intelligence. With greater adoption of AI, the current practice might cause loads on the electricity grid that are unsustainable.

One method that can alleviate the problem is "demand shifting". This means that workloads can be run at places and times where electricity is currently cheap and generated by sustainable sources. For example, one could postpone the (costly) calibration of some AI model if the wind speed is currently low by a few days/weeks, if one expects a change in weather. This could be supported by forecasting algorithms that can predict the weather and/or the load on the power grid. Likely, there is a lot that AI/ML could bring here; a supervised algorithm could predict the spot price of electricity based on weather data, current load, season, etc.

2.3 Regulation and compliance

Since 2016, there is an extensive framework of regulation concerning data privacy and security, called GDPR, in place in the European union. GDPR sets conditions for what is considered personal information, and how it must be handled. GDPR is generally formulated in vague terms such as "taking appropriate meausures". This is by design since it is meant to be applied to so many different technologies, in a rapidly changing landscape. What "appropriate" means, will be up to a court to decide, on a case specific basis.

One of the requirements of GDPR is that a data controller needs a valid reason to process personal data. After the data is no longer needed it also needs to be erased. In machine learning, personal data may be used to train models. In that case, it is important that the full dataset of personal information is stored by the controller wherever the training is happening. The data can (must) then be locally be deleted from that server, but the question is whether the controller may store it for future use somewhere? After all, machine learning models are typically retrained after some time. One solution is to anonymize the data, i.e., make it so that no data point can be identified. If this is impossible, the controller must make a judgment of whether retaining the data qualifies as necessary, and proceed under the legal pressure that a court might not agree. Therefore, it is important that machine learning developers are well versed in such judgments.

3 Future trends

Machine learning (ML) is growing into an ever more important engineering tool. As it gets used for more complex tasks, it runs into the same problems as traditional software engineering (SE) regarding complexity, maintenance, versioning etc. For that reason, machine learning needs to draw on the preexisting toolbox for handling these, while also developing its own novel, more ML-specific, mechanisms. Thus machine learning and software engineering will likely be moving closer, especially as some SE-tasks (e.g., code prediction) may be best handled by ML.

However, I do not see the fields as merging. Machine learning is a field that fundamentally lies close to mathematics and information theory. The fundamental theory does not depend so much on software, but pen and paper equations. Implementing it is also usually possible with only rudimentary numerical libraries. Software engineering only needs to come into the picture when the projects get more comprehensive. At the other end, ML has the potential to do things that conventional software is not close to, for example photorealistic art-generation. Its full potential is far from known and can likely only be (partially) discovered by human experimentation. It is such a young field that it is even hard to state what its own goals should be.

On the other hand, SE is more about helping humans build reliable, working systems as easily as possible. This is more of an ergonomic endeavor than a scientific, or mathematical one. Thus, the goals of ML is not really aligned with those of SE. On the other hand, it remains to see how much software engineering can draw from AI/ML. Possibly, autonomous coding could become so good that human input reduces to just providing instructions for what is to be achieved by the final product. Currently there is a lot of progress in the field but it is far from providing complete programs for an end user. As earlier mentioned, software engineering is already using machine learning at great benefit for automatically generating small snippets of code. There is no reason not to believe that this usage will expand, and predict larger sections of code. Possibly, the burden of the programmer can be focused more on features to be included than manually wiring them together. However, this will likely still be far from any kind of panacea of self-writing software.

All in all, machine learning and software engineering will absolutely keep benefiting from each other while cross-pollinating. Still, fundamentally they are different and should stay true to their own main goals for the time being.