Software Engineering Assignment

Max Nilsson

August 2023

Introduction to my research area

My research area is mainly about theoretical improvements in modern optimization theory, and in particular how these can be applied in nonlinear and nonconvex deep learning settings.

An underlying assumption in much of optimization is convexity. Convexity is a way of connecting local properties of an objective function (such as its gradient) to global properties (such as its minimum value). That way, if we can solve some problem involving its gradient, then we can for instance minimize the objective function. This is done by using an appropriate algorithm.

One such algorithm is the classical Newton method, which utilizes what is called second order information. In many contexts, the Newton method works very well, but a problem occurs when we consider its computational complexity. For huge problems, such as optimizing some deep neural network, even performing one iteration of the Newton method becomes infeasable. What we are left with are the first order method, and my research concerns one such class of methods called the *Bregman methods*.

A Bregman scheme furthermore uses the geometry of the objective function in a clever way, so that we can solve problems of wider class than those solved by the normal Gradient Descent method. In fact, the Bregman Descent method is a strict and meaningful generalization of the Gradient Descent. There seems to be a very intriguing connection between which Bregman scheme used, and the generalizability of the trained neural network. Exploring this connection is the long term goal of my PhD.

My current research is a project concerning the *symmetry coefficient* of a Bregman Distance. The Bregman Distance is something used in the update step of the algorithm, and when this distance is the standard 2-norm, we get the Gradient Descent.

One aspect of a general Bregman Distance is the fact that it is non-symmetric. This is both a weakness and a strength. It makes the analysis of the algorithm harder, but it is just this asymmetry that makes it possible to efficiently perform the algorithm in the first place. But, under certain conditions, we can still guarantee the convergence of a Bregman scheme, if our step-size in the algorithm is below some number which depends on the symmetry coefficient. In other words, if we knew this coefficient we could take more aggressive step sizes in the algorithm, and hence maybe converge faster.

Two Concepts from Robert's Lectures

Two concepts stuck with me from Robert's lectures. The first is the topic of training machine learning invariants, which can be detected with the use of metamorphic testing. This concept is interesting both in its ethical implications and its theoretical ones. In some sense, machine learning models in particular learn biases in the training data, and will use these biases to its advantage in getting a better evaluation score. We have seen many examples of this backfiring in real world examples, leading to sometimes hard ethical

considerations. But it is also a question about model quality. For instance, we always want our cancer classifier to perform well over different countries around the world.

How to actually implement these machine learning invariants seem to be a fascinating theoretical problem. During this years ICML I watched a very inspiring presentation about this and how they formulated the training as a min/max-problem (similar to the GAN training framework).

The second concept is the life cycle of deep learning models. We would like the learning of deep neural networks to work as an iterative piece of code, so that we can easily implement changes in our network. But unlike pieces of code, which are hopefully constrained in some scope of files, changing a small hyperparameter in a neural network will impact a huge number of weights. Furthermore, it might not impact enough, since we would be stuck in a local minima of the loss landscape. In such cases it might have been advantageous to begin the training from an earlier point. From what I gather, this fine tuning is one of the reasons behind the success of ChatGPT, and so it seems like this is a very important point.

I will return to this point in the last section, where I will briefly discuss the long term life cycle of deep learning models.

Two Ideas from Guest Lectures - Sentiment Analysis

I found this lecture to be an inspiring example of an application of machine learning. The first idea I would like to bring up is the combination of feature extraction/selection with choosing a suitable machine learning model/algorithm. I have taken a course in natural language processing and I recognize some of the concepts, but I do not know what the *role of negotiations* refers to. Either way, both of these steps are essential to training any neural network and is at the fundamental core of the applications of my research. It is also intriguing how convolutional neural networks can be used in a setting which is seemingly unrelated from its standard image analysis application.

The second idea is that of the software engineering specific sentiment analysis research and its specific problems. It shows that even if a model works really well in one domain, it might collapse in many cases in other domains (e.g. wikipedia vs. twitter). These missclassifications are natural for us humans, but not always for the model. My thoughts are either that the underlying model works at a too local scope, which may need to increase. Furthermore, the model might need to incorporate some kind first order logic/semantic system which would help it classify tricky cases. This latter is probably much harder to implement, while still keeping the fast classification that the model has.

Two Topics from the List

As with any new technology, in this case the emergence of efficient machine learning methods, they will propose solutions to existing problems while also creating new issues and problems. In this section, I will briefly cover my thoughts how machine learning relates in these aspects to the sustainability of software/developers and automated software testing.

Sustainability

Sustainability is usually described in three main areas: economic, social and environmental. In the modern machine learning industry these are in part a subset of the sustainability analysis of normal programming environments. For instance, it makes economic sense to write clean and reusable code, which usually follows classical design patterns. The Agile framework is probably the most famous one. Here it is said that code should be written parallel to test driven scrutiny. This also ensures that the code is deployable, which usually works in favor of the social sustainability. With a greater emphasis on communication with the client, it leads to better communication with the individual coders and a better work environment. Furthermore, there are beneficial social concepts such as pair programming and stand up meeting. The environmental aspect is in my opinion often underplayed here, but I know of *green code* and the concept of writing code which brings less harm to the environment.

The machine learning sector brings some unique areas of opportunity and problems. We have all seen the ridiculous numbers which how long it takes to train the biggest neural networks today (for instance, chatGPT). The power consumption is enormous - utilizing massive data centers - and the implications on the environment become much more grave. How can the big companies responsibly train large models with respect to the environment? How can companies risk training multiple versions of a model when the energy bill runs so high? In many ways, this is connected to my research. If our algorithms were more optimized and yielded networks with greater generalizability, then hopefully this would impact these problems. The issue is that it is not necessarily true that companies will train less, even if there algorithms were better, since the algorithms will hopefully be open to everyone - including their competitors.

On a more positive note, we have already seen how machine learning models can be used to software products and engineers. They can for instance help write more sustainable code, saving time for the developers and money for the companies. They are also used to quickly scan your code and find ways to improve it, often refactoring to established design patterns. One of the main problems that come with these use cases are the interpretability of the code, see [2]. Machine learning models are black-box models by nature, and it is also the main reason why they are capable of being so strong, but the downside is that, for a vanilla neural network, we will not get any answers to *why* the output to a model is the output. In [2] they cover some interesting results using stochastic symbolic expressions

in combination of genetic algorithms to achieve neural networks that can in a sense give explainability of their outputs. These concepts are new to me, but I believe this problem will surely impact my research in the coming years.

Automated Software Testing

The importance of automated software testing cannot be understated. I believe that any serious piece of code could not get to the place it is without successful testing. In some sense, programmers do this all the time when writing ASSERT statements in their code. But the automated software testing in the Agile framework uses a much wider class of tests, such as functional unit testing and security testing. A substantial amount of time of a substantial amount of programmers at large tech companies is spent on writing test cases. If we could find a way to automize the writing of these cases, perhaps using machine learning models, this would greatly impact companies economically.

We have seen recently some developments on this front. Not only can code writers such as GitHub Copilot automate test cases, but there is some academic advances also, such as in [1]. Here they trained a neural network using ant colony optimization and got good results when trying to detect bugs in code. To close the loop, you would then want some language model to write the corresponding test code. I have before encountered ant colony optimization during my time in industry, but I have not yet have had the opportunity to see how it can be connected to the training of neural networks (which would make it very relevant to my research).

Discussion of Future Trends

I have tried to form my previous discussion points around the parts which I think will have a big impact on the future trends. For one, I think automated machine learning will grow and play a role in our society. *Solving a task* in the future might be equivalent with *finding the correct data/model and training a small network*. If so, I think it is interesting to reflect on how efficient and good these general automatic model will be. One could imagine that to meet all demands, the network will not be particularly specialized at anything, yielding a crises in quality.

I also think that training machine learning invariants and getting models that are explainable will be a great theoretical and practical challenge.

Every issue that machine learning models have from a software engineering point of view, will be amplified when considering not just one singular neural network, but a whole system of them. Since the explosion of ChatGPT, multiple variants of large language models have appeared. It is not difficult to imagine that our future will entail multiple daily interactions with different machine learning systems, each of which using multiple collaborating neural network.

A final aspect I have considered when writing this assignment, is what will happen with some specific neural network in the global time scale. Will anyone use ChatGPT in 100 years? Probably not. But perhaps the superior models then will somehow be built upon the work of ChatGPT (such as a more extreme version of fine tuning it or using it as a pretrained network). Maybe future neural networks will reach a limit on how many neurons can be reasonable used, and instead emphasise some sort of ensemble training of multiple, different neural networks.

References

[1] Subarna Shakya and Smys Smys. "Reliable automated software testing through hybrid optimization algorithm". In: *Journal of Ubiquitous computing and communication technologies (UCCT)* 2.03 (2020), pp. 126–135.

[2] Ricardo Vinuesa and Beril Sirmacek. "Interpretable deep-learning models to help achieve the Sustainable Development Goals". In: *Nature Machine Intelligence* 3.11 (2021), pp. 926–926.