Assignment
WASP Software Engineering Course Module 2023

Adam Orucu

Introduction

My topic is to research Neural Architecture Search (NAS) and how it can be applied to improve Transfer Learning (TL). NAS is a class of methods that provides an automated way to find optimal neural architectures for a given task. It usually uses Reinforcement Learning or Evolutionary search methods to explore the search space of possible architectures without training and testing every single architecture in the search space. TL is a group of methods that use models that were trained on different tasks or data as a starting point for a new task or data therefore greatly speeding up the training process and in cases where the data is limited its prediction performance.

Concepts from Robert's lecture

Concept #1: Test Driven Development (TDD)

Given the inherent complexity in NAS – from designing a nuanced search space to creating an accurate representation of a selected architecture – there is ample room for error. Implementing TDD ensures the robustness of each layer in the development process. For example, tests can validate the breadth and precision of the search space, ensuring no invalid architecture configurations arise. Then, other sets of tests may check if the selected architectures create and train ML models correctly or if the search strategy works as intended. Furthermore, in the context of Transfer Learning, tests can be created to confirm that pre-trained layers and weights are coppied correctly. Of course thinking about general machine learning application, one can also test if repeat runs with the same seeds result in same dataset splits or performance score.

Concept #2: Metamorphic testing

A quite applicable example of metamotphic testing for NAS, is checking if the best architecture in a large search space is also the best architecture in a smaller search space where the same architecture exists. This could be used to test the evaluation metric that is used in the method. To check if the performance of an architecture is same whichever search space it is in. However, more relevantly it could be used to test the search strategy. If the search method is able to find the best architecture in a large search space, it should be able to find it in a smaller search space that contains the same architecture. Similar tests already exist for transfer learning where models trained only on source and only or target domain data are used as baselines specifying, the minimum and maximum expected performance respectively. This means that the transferred model should be in-between these scores.

Concept #3: Code versioning

Version control is an obvious thing to use in ML software development. It might be even more useful to a researcher working on NAS. First, because as a researcher I have to test different methods and changes to different methods which is easier and safer using version control. Additionally, because NAS consists of different replaceable parts, for example the search strategy (e.g. RL-based, evolution, random search) it is nice to have submodules that can be replaced or rolled back at wish.

Concepts from guest lecture

Concept #1: AutoML

Automated Machine Learning (AutoML) and Neural Architecture Search (NAS) are both advanced methodologies aimed at simplifying, optimizing and automating the machine learning model development process. While AutoML covers a broad spectrum of tasks, from data preprocessing to model selection and hyperparameter tuning, NAS specifically focuses on finding the best neural network architecture for a given task. Essentially, NAS can be viewed as a specialized component within the larger AutoML framework, dedicated to automating the typically labor-intensive and expertise-driven process of neural network design.

NAS consists of three main parts; search space, search strategy and performance evaluation. Search space options that are selected by the user which define all the possible/allowed architectures. The search strategy is used to sample architectures from this search space and learn from its selections to pick better architectures overtime. Lastly, the performance evaluator evaluates the architectures that the search strategy picks.

One could also consider a bit more meta application of AutoML to my topic. Many NAS methods use other machine learning methods within them which select the architectures. AutoML techniques could be used to find best configurations for this models as well.

Concept #2: Boundary value testing

By applying Boundary Value Testing, researchers can ensure that the boundaries of the search space are correctly and robustly defined. For instance, if a certain neural architecture parameter can vary between specific limits (like the number of layers in a network or the number of nodes in a layer), BVT can be used to test architectures at those minimum and maximum values, as well as just outside them to check for system robustness and stability. This helps in avoiding potential pitfalls like overflows, underflows, or other unintended behaviors that might compromise the architecture search process or the performance of the resulting models. One other thing that could perhaps be categorised as boundary value testing is training tiny neural networks or training for 1 epoch. In such cases the models might simply predict the most common class. In cases where the classes are inbalanced this might even get higher scores than larger neural networks which is ofcourse not desirable and might help in finding better solutions.

Concepts from list

Concept #1: Automated software testing

Automated software testing is the process of running pre-scripted tests to ensure functionality and performance. This could be done with unit tests, testing small parts of the software at a time or the whole program at once. The benefits of this type of testing is that it can be run frequently - for example every time a change in codebase is made. It can also help in getting consistent results since the same script is repeatedly run and there is no deviation that could otherwise be incurred by human testing. One interesting example of this kind of testing can be seen in CI/CD where specified test may run automatically every time a commit is made or code change is pushed to the server.

Some tests that can be applied for machine learning have been listed in previous sections - unit testing, metamorphic testing. These tests could be automatised to spot errors faster and develop more stable systems. There are, also, a couple of ideas on automated testing of machine learning software in [1]. An interesting one is using linters to detect errors in datasets. Some other methods generate synthetic data to test the neural nets, others integrate numerical-based tests.

In general automated testing could be considered as part of NAS. In the performance evaluation part of the NAS process some methods put additional restrictions which might aim to restrict the neural network in terms of memory or computation limits on the target device. In such a case the neural network would need to be tested on a device whether it breaks those restrictions.

One other idea could be utilising automated tests in completely automatised machine learning pipelines perhaps similar to the software tested in the lectures. Where many different ML methods and different DL architectures (using NAS and TL) are created and automated tests are used to check if they are acceptable.

Concept #2: Maintenance and evolution

Maintenance and development refers to the changes that are continued to be done after the initial development of a system. In reality the environment a software is deployed into change overtime, therefore previously working software may end up not working after a while. Similarly the requirements of the users might change overtime, in such a case the software needs to be evolved to fit the users expectations.

The way this is reflected in machine learning systems is quite similar. While changes in the user requirements or the software environment would affect a machine learning system similarly to a regular software system one difference that is machine learning specific is related to the collected data. The data collected during from the system that is used to make predictions may change overtime. This difference is usually data-shift which is a change in the underlying distribution of the data. The number or type of features could also change. In both of these cases the previously well performing models would perform worse over time.

In [2] the authors specify four software engineering challenges in maintaining large ML systems; adaptability, scalability, privacy and safety. As previously mentioned, one thing that needs to be kept in mind to ensure adaptability is looking at the data acquisition process and see if the changes have happened since the last training. Additionally, according to [2] feature engineering is important to track for adaptability. It is might be useful in changeing the raw data to still work on the model without the need to retrain. The paper also mentions transfer learning in relation to scalability

As the field of deep learning is rapidly advancing, the architectures that are optimal today may not retain their superiority tomorrow. Thus, the NAS algorithms and systems require regular maintenance to ensure they function seamlessly with new datasets, hardware, or frameworks. Furthermore, as new techniques or insights emerge in neural network design, NAS systems need to evolve to incorporate these advancements into their search paradigms. Transfer Learning models, given their nature of leveraging pre-trained knowledge, might need evolution to integrate newer, more effective pre-trained models or to adapt to shifts in source or target domain characteristics. Regular maintenance ensures that these models continue to transfer knowledge effectively despite shifts or drifts in the underlying data distributions. In summary, recognizing the necessity of maintenance and evolution in the context of NAS and TL is essential for sustained success and adaptability in an ever-changing technological landscape.

Future

Application that is in development and hopeful will excelerate is adding NAS to the AI development workflow. Currently these workflows are quite simple; create dataset, set a network and test. The problem occurs when the developer doesn't know what size or shape the neural network should be. It is easy to overshoot unnecesarily creating a huge network wasting a lot of computation or undershoot receiving underwellming results. Adding NAS and/or TL to this process might be really beneficial.

This adds an abstraction to the development of a neural network which is both good and bad. Ofcourse it is good in the way that it might ease the development while improving performance, but an additional abstraction may make neural networks hard to understand which currently already are black boxes. Advances in behavioural software engineering might become really useful in this aspect. It also relates to hidden technical debt mentioned in the lectures.

Lastly, looking at the further future it is impossible not to mention LLMs and their impact on software development. They have already change the way I write code and will definatelly change even more. One thing I realised is that I've started creating bugs that are harder to find. This is something I and others will need to be mindful of. It also clearly shows the importance of testing, especially unit testing, to check if everything behaves as expected.

References

[1] H. B. Braiek and F. Khomh, “On testing machine learning programs,” *Journal of Systems and Software*, vol. 164, p. 110 542, 2020, ISSN: 0164-1212. DOI: `https://doi.org/10.1016/j.jss.2020.110542`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0164121220300248`.

[2] L. E. Lwakatare, A. Raj, I. Crnkovic, J. Bosch, and H. H. Olsson, “Large-scale machine learning systems in real-world industrial settings: A review of challenges and solutions,” *Information and Software Technology*, vol. 127, p. 106 368, 2020, ISSN: 0950-5849. DOI: `https://doi.org/10.1016/j.infsof.2020.106368`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0950584920301373`.