WASP Software Engineering Assignment

Shivam Mehta

August 15, 2023

About me

I am Shivam Mehta a PhD student at KTH Royal Institute of Technology, my research is focused on probabilistic generative models for text-to-speech synthesis. I train large deep learning models with generally millions of parameters to synthesise ”human-like” speech. It involves developing different deep-learning models with the intersection of probabilistic interpretation to estimate a distribution over complex speech acoustics, which later can be used to sample from and generate different natural-sounding speech samples. After creating these models, I investigate new training and inference methods in an effort to enhance their performance. For instance, I'm currently working on creating new algorithms for using enormous datasets to train these models, which can be computationally demanding work. In order to make these models more expressive, I'm also looking into how to add more elements like prosody and emotion to these. My research focuses on probabilistic machine learning as it is an incredibly valuable approach when it comes to modeling the distribution of data within a given training dataset. It allows us to effectively capture the uncertainty and variability inherent in complex datasets. In the context of my research, I am interested in exploring the potential of probabilistic machine learning in conjunction with human speech and human motion data. The goal is to leverage these multimodal data sources for applications such as text-to-speech synthesis and the generation of nonverbal gestures to accompany synthesized speech. By seamlessly integrating these modalities, I aim to push the boundaries of human-computer interaction and facilitate more natural and immersive interactions between humans and machines.

My research also includes assessing the effectiveness of these models through user research and unbiased measures. I aim to find opportunities for further development and create fresh approaches to these problems by evaluating the advantages and disadvantages of these models. Recently, we introduced generative models like neural HMMs [1] and a fusion of neural HMMs and Normalizing flows named OverFlow [2], to synthesise speech and to model the complex distribution of speech acoustic. These are autoregressive, probabilistic neural text-to-speech models which generate 'human-like' speech. These models require iterative development and each version is tracked during their development lifecycle.

Concepts from lectures

In my deep learning research, I place significant emphasis on incorporating software testing and software maintenance as fundamental principles. One key practice that I personally adopt is **Test Driven Development** (TDD), wherein the benefits of having a well-structured test suite far surpass the initial time investment. Implementing TDD not only enhances the modularity of my package but also facilitates seamless maintenance as I continually update my source code to accommodate various datasets, diverse use cases, and the addition of new functionalities to the system.

Having a robust test suite is especially advantageous when it comes to bug detection and issue resolution. As my research progresses, I often encounter new bugs and challenges. Thus, the presence of a comprehensive test suite allows me to promptly identify and resolve these issues, ensuring the stability and reliability of my deep learning models. Furthermore, software maintenance becomes increasingly significant when user requirements evolve over time which is rarely in research but I often change my research hypothesis or try different datasets then this becomes significantly crucial. By actively maintaining the software, I can adapt my models to meet these changing needs,

[1] https://shivammehta25.github.io/Neural-HMM/
[2] https://shivammehta25.github.io/OverFlow/

ensuring their continued relevance and effectiveness in real-world applications. As datasets vary, and new use cases emerge, software maintenance allows me, along with other researchers, to flexibly update and fine-tune the models. Furthermore, the use of Github actions to manage the **AI/ML process/workflow** allows me to not to worry about testing this on my machine but I rather let the test suite run on the CI/CD pipeline.

From the guest lectures

Dev bots are valuable tools that existed prior to the AI/ML-powered bots. Before the introduction of GitHub Copilot, I used various linters (sonar for Java, Mypy for Python) and formatters (black, flake8 for Python). These tools significantly enhance my workflow by managing various overheads and ensuring the written code function as intended. These days the development of GitHub copilot has further improved my code writing process. When faced with the challenge of writing a trivial function, I often describe my requirement in a comment, retrieving a base code and modifying it further to suit my preference. These AI-powered tools are also very useful in writing documentation. However, they lack the capability to personalise and configure, which rule-based devbots excel at. Therefore, instead of replacing the rule-based devbots I find it more advantageous to fuse them together where I prefer my customised configuration to the rule-based and use Copilot in writing the code as I write. Another interesting tool I find very useful is pre-commit hook [3] This helps me to run these rule-based devbots right when I commit and also if someone else pushes the code to my repository, it automatically formats the source code to match my style guide. Another presentation that instantly got my attention was about **AutoML**. AutoML sounds interesting as it removes several boilerplate bottlenecks and allows the engineer to quickly test their datasets against the leading machine learning models. However, while being convenient for an engineer, it might hinder the customization that can be achieved for a researcher. I feel as a researcher striking the right balance between AutoML or boilerplate-free tools and domain knowledge and customisation is essential to harnessing the true potential of AutoML and for it to be able to help me effeciently.

Topics from the list

Automated Software Testing

"Testing" is not specific to the field of software development; most products, once developed, go through a testing phase. One major benefit we have in the field of software engineering is that this phase can be very efficiently automated with the help of modern testing tools like JUnit for Java, unittests and pytest for Python, etc.

As a researcher working with deep learning, I find automated software testing to be very important for developing my research prototypes. Deep learning models are complex, and their behaviour can be difficult to predict, making testing a crucial step in ensuring my work is reliable and effective. Traditional manual testing methods are often time-consuming and can have errors, especially when dealing with large and intricate neural networks. Automated software testing helps me efficiently check the performance and reliability of my deep-learning models and makes sure that the model is robust to adversarial examples or even some other unexpected inputs [MPL19].

One main benefit of automated testing is that I can run many tests repeatedly and consistently. I can create test suites that cover various scenarios, edge cases, and data distributions. This extensive testing helps me find and fix potential issues early in the development process, leading to more stable research prototypes. Additionally, I can easily integrate automated tests into my continuous integration and deployment processes, ensuring that any changes or updates are immediately evaluated for their impact on model performance [KHL19].

In my own model development, I try to incorporate different testing paradigms, from unit testing of functions to integration testing of other components together, I wrote the whole test suite of my model[4] with the help of pytest, during the development I prefer to use Test Driven Development, this helps me to catch bugs early when I

[3] https://pre-commit.com/

[4] https://github.com/shivammehta25/OverFlow/tree/main/tests

make modifications to the architecture of my system, it is especially notorious with deep learning systems where the inputs to functions are tensors with some dimensions and the order for these dimension matters

Moreover, automated software testing gives me valuable insights into the model's behavior and possible weaknesses. I can use techniques like adversarial testing, boundary value inputs or, where the model is exposed to carefully crafted inputs designed to exploit vulnerabilities. This helps me understand the model's failure modes and areas that need further improvement. The feedback from automated testing helps me to develop faster and more reliably, resulting in more robust research prototypes. Personally, the most crucial use case I find during the development lifecycle is the ability to expose regression issues very early, when running the test suite I easily see that these issues have arisen because of my recent changes or commits, and then I can either modify and fix them or roll back and modify my code again. This provides a systematic and transparent way to document the testing process, making it easier for other researchers to understand and reproduce the experiments. Reproducibility is essential for validating and building upon previous work especially in the field of research and development.

In conclusion, automated software testing plays a crucial role in my work as a researcher in deep learning. It enhances the development of research prototypes by providing consistent and comprehensive evaluation, optimizing hyperparameters, and revealing possible weaknesses. The efficiency and accuracy of automated testing enable me to focus on pushing the boundaries of deep learning research, leading to more innovative and impactful advancements in the field.

Continuous Deployment

Continuous Deployment (CD) in a software development lifecycle generally facilitates the rapid and efficient delivery to production systems, while as a PhD student, I rarely have encountered the need to push my research prototypes into any production system. But, one important advantage of Continuous Deployment in research is that it allows me to test new model versions with real-world data rapidly. As I try different architectures, hyperparameters, and training strategies, deploying updated models continuously lets me see how they perform in the actual environment. This way, we can find and fix problems faster. This includes running tests on the CI/CD pipeline. While developing code I regularly test my pushes/pull requests through CI/CD pipelines via GitHub actions. [5]

Continuous Deployment also encourages collaboration between researchers and developers, promoting a culture of continuous improvement. By deploying new models and features regularly, we can work together more effectively, find issues faster, and solve problems quickly. Automated testing seamlessly integrates with continuous integration and deployment pipelines. This enables us to automatically test models with each code change or update, ensuring that only stable and well-performing models are deployed to production.

This especially can be seen in open-source machine learning development. Where major companies like HuggingFace [6] are frontier in the development of open-source alternatives for commercial machine learning systems. Continuous Deployment helps them to adopt best practices in developing and testing deep learning models. Models and their code are automatically deployed and tested in environments similar to production, so they can identify and fix issues before they affect users. This ensures that their models are of high quality and provide a good user experience [Ren+19] especially while developing with a large number of contributors. However, we must keep in mind that this validation and testing should be robust and only stable releases should make it past the pipeline and we must have measures in place to fall back to a past different release in case something buggy gets deployed.

It is also useful for building various deep learning research pipelines, these pipelines also include data reprocessing, model architecture definition, hyperparameter tuning, and training process automation. We can set up the data preprocessing pipeline and configure the hyperparameters in a way that we can in the pipeline run a hyperparameter sweep and given objective metrics we can simply save our research time and do it on the continuous deployment pipelines. I believe major companies use CD extensively as they need to continuously update their models with the stream of new data that they acquire from their users. Therefore, the use of Continuous deployment leads to

[5] https://github.com/shivammehta25/OverFlow/actions/workflows/test.yml
[6] https://huggingface.co/

faster iterations, improved collaboration and higher quality models and should be incorporated even in a research and development lifecycle.

Future trends and directions of Software Engineering

I believe that AI/ML Engineering is on an upward trajectory, with more and more use cases unfolding. In the current landscape where the demand for open-source architectures and frameworks are of utmost importance, the demand for software engineering will rise as the models become more and more complex. This is specially true when we consider the variety of uses cases that ML is emerging pioneer in. In the upcoming 5-10 years, I believe the biggest boost will be with devbots with large language models improving and reducing the gap between human and machine comprehension. A new paradigm of AI model lifecycle management will become a bit more prominent than what it is today. Further, I believe more detailed software might emerge that focus on ethical considerations, fairness and explainability of these AI models.

References

[KHL19] Divyansh Kaushik, Eduard Hovy, and Zachary C Lipton. "Learning the difference that makes a difference with counterfactually-augmented data". In: *arXiv preprint arXiv:1909.12434* (2019).

[MPL19] R Thomas McCoy, Ellie Pavlick, and Tal Linzen. "Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference". In: *arXiv preprint arXiv:1902.01007* (2019).

[Ren+19] Cedric Renggli et al. "Continuous integration of machine learning models with ease. ml/ci: Towards a rigorous yet practical treatment". In: *Proceedings of Machine Learning and Systems* 1 (2019), pp. 322–333.