Research Statement: Jeevana Priya Inala

My research agenda is to combine machine learning and program synthesis to build robust intelligent systems. Traditional deep learning has successfully solved many challenging problems in artificial intelligence (AI), but it has several limitations that affect its usability–e.g., it is data-hungry, produces opaque and brittle models, and lacks any guarantees. **My research** alleviates the limitations in deep learning by incorporating **program** structure (such as loops, conditionals, and list operations) in the deep neural models. The rich symbolic structure in programs allows us to get **interpretable**, **generalizable** and **robust** models. Program synthesis—a field dedicated to automatically generating programs—has had tremendous success in the software domain. However, these program synthesis techniques have mostly focused on synthesizing discrete programs, leveraging well-known discrete search algorithms such as SAT solving and enumeration search. However, in the domain of intelligent systems such as robots, we need programs that have both discrete and continuous components. Hence, I propose a **neurosymbolic learning** approach that combines traditional machine learning, numerical optimization, and discrete search algorithms. The neurosymbolic learning approach is applicable in a wide range of domains; in particular, my focus is **robotics** where reliability, robustness, and ability to enforce constraints are essential for safety. I focused on three different questions related to this approach:

- **Generalizability and Interpretability:** Can we learn control policies for robots from a few examples, and generalize to arbitrary sets of new examples?
- **Combinatorial constraints:** Can we enforce combinatorial constraints regarding the structural and the resource limitations in the robots, specifically in the context of multi-agent communication?
- **Compositionality:** Can we synthesize how to compose reusable modular components to build robots that do different tasks?

Below, I first describe my research on answering the above questions; then, I outline the future research directions.

Learning generalizable robots

Before I started thinking about generalizable robots, I worked on several program synthesis projects such as OptCNF [6] that synthesizes programs to generate domain-specific bit-vector transformations code in SMT (satisfiability modulo theory) solvers, and WebRelate [5], a project done during an internship at Microsoft Research, that synthesizes programs to combine data from spreadsheets with unstructured data from the web using examples as the specification. The intriguing aspect of these projects is the generalization ability of the synthesized programs. For example, for the bit-vector transformations, while we synthesized the programs to work on bit-vectors of length ≤ 3, the programs also produce the correct transformations for bit-vectors of length 32 (that are actually used in the SMT solvers). Similarly, the WebRelate tool can synthesize a correct program from just 1 or 2 input-output examples and yet generalize to all the inputs in the spreadsheet.

Generalization is also important in robots. Autonomous systems must possess the capacity to work in various environments, including environments never seen previously. To achieve this, autonomous systems have to learn to extrapolate from the scenarios seen before. For e.g., consider the autonomous car (blue) in Figure 1, whose goal is to move out of a parallel parking spot. From the first three scenarios, it is clear that the car needs to do repetitive back-and-forth motions to exit the parking spot. Now, we can extend this logic to scenarios where the gap between the cars is tiny (something that has not been encountered before).

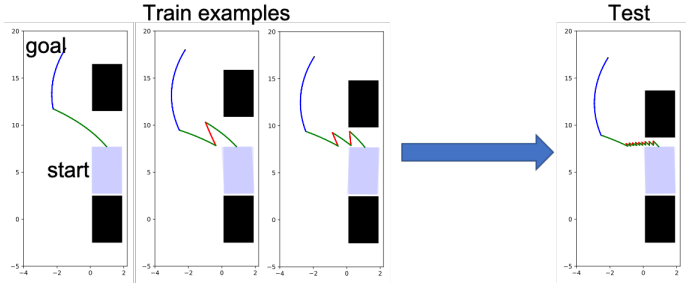

Figure 1: An example of an autonomous car driving out of a parallel parked spot.

To learn such generalizable behaviors, I cast the control learning problem as a program synthesis problem [3]. In my approach, program control policies have loops and conditionals that can capture the repetitive structures necessary for extrapolation. Additionally, these program policies are interpretable and can be modified post-learning to get different

behaviors. On a set of reinforcement learning (RL) tasks involving quadcopters and cars, I have shown that while traditional deep RL approaches perform well on the original task; they fail to generalize. In contrast, our program policies successfully generalize beyond the training distribution.

Learning program policies is challenging because of two reasons: (1) control programs have both discrete and continuous structures, and hence, we can not use traditional program synthesis algorithms like SAT solving and enumeration that work best on only discrete problems; (2) programs have very few parameters which make gradient-based optimization techniques run into local optima. To address these issues, I proposed an approach called adaptive teaching that alternatingly learns a teacher and a student. The teacher is an over-parameterized neural-like version of the student, and the student is a program policy trained to mimic the teacher, hence the name neurosymbolic approach. The teacher does not generalize as well as the student, but because the teacher is over-parameterized, it can be easily learned using traditional RL algorithms. Furthermore, in my approach, the teacher is regularized to favor strategies similar to the ones taken by the student, to ensure the student can successfully mimic the teacher. As the student improves, the teacher improves as well.

This work is the first to show that it is possible to learn programs with loops for reinforcement learning problems. Our work further illustrates that learning programs is as beneficial in the robotics domain as it is in the software domain. Finally, the concept of using an over-parameterized neural-like intermediate to help guide the search for programs generally applies to many settings that require synthesizing discrete-continuous programs (as seen below).

Enforcing combinatorial constraints in robots

To learn the design and control for robots, we have to satisfy the resource limitations of the physical hardware. This problem is especially important in decentralized multi-agent planning domains, where the agents have to communicate with other agents to coordinate their actions. Here, the goal is to learn how to coordinate with other agents, both deciding whom to communicate with and what information to share, and at the same time, minimizing the amount of communication required to satisfy the limited bandwidth constraint. This minimum communication constraint is a combinatorial constraint which makes training neural network policies hard. On the other hand, programs are easily amenable to combinatorial optimization.

Based on these observations, in collaboration with a group of researchers at MIT and UPenn, I proposed to learn multi-agent communication policies as programs in a domain-specific language (DSL) [7]. Our DSL includes components such as filter, map, and random choice that operate over sets of inputs, because choosing whom to communicate with requires reasoning over sets of other agents—e.g., to avoid collisions, an agent must communicate with its nearest neighbor in its direction of travel. To learn these programs, similar to the previous project, we leverage an over-parameterized intermediate representation to guide the search for programs. In this case, we use a transformer policy as the over-parameterized neural intermediate. The transformer policy is trained without the combinatorial constraint. Then, we use a program communication policy to replace the transformer's soft attention with hard rules, forming a neurosymbolic transformer. The program policy is trained to optimize two goals: (i) match the transformer as closely as possible, and (ii) minimize the number of communications at each step.

We successfully used the neurosymbolic transformer approach on several multi-agent planning tasks that require agents to coordinate to achieve their goals. Our algorithm learns communication policies that achieve task performance similar to the original transformer policy (i.e., where each agent communicates with every other agent), while significantly reducing the amount of communication.

This work shows that neurosymbolic transformers are promising for training policies that additionally optimize combinatorial properties on graphs. The ideas in this work could potentially impact other applications such as natural language processing (NLP), where transformers are state-of-the-art. Our work is also a step in achieving interpretable and generalizable transformers by replacing soft attention weights with programmatic attention rules.

Building robots using modular components

Just like how software is composed of reusable modular parts such as libraries, modular robots allow us to build on-demand robots from modular components. However, choosing the "right" design based on a task is a challenging problem because the design space contains both discrete and continuous parameters. The challenges in learning the design are similar to the challenges in learning the program policies in the previous research directions. Therefore, in collaboration with Hadas Kress-Gazit and a graduate student at Cornell, we came up with a synthesis algorithm to solve the modular robot design problem [1]. We realized that a straight-forward approach of searching directly in the design space leads to a highly non-linear search space, which gives rise to numerous local optima. Instead, we proposed an alternate approach of searching in an over-parameterized space, which makes it less susceptible to local optima. In

addition, our approach can synthesize design(s) for partially infeasible tasks. Suppose a single design cannot solve the entire task; our approach can either separate the task into feasible and infeasible portions for one best design or find multiple designs that together can achieve the whole task.

Besides the above-mentioned works, I have also worked on some other projects that further illustrate the impact of programs in building AI systems. In collaboration with a graphics group at MIT led by Wojciech Matusik, we combined program synthesis with geometric processing to reverse-engineer the process by which 3D models may have been generated [2]. Our method learns parameterized constructive solid geometry (CSG) programs, providing a powerful means for end-users to edit and understand the structure of 3D models. In another work [4], in collaboration with Osbert Bastani at UPenn, we ensure safety modulo fault for human-interactive robotics systems by using a simple program shield policy together with on-the-fly abstraction/verification to override an arbitrary controller.

Future directions

I am excited about developing more neurosymbolic techniques for learning robust AI systems. In addition to extending and building on previous systems, I plan to explore the following ideas.

Applications: Within robotics, there are several other applications where I want to apply the neurosymbolic learning approach. Perception and dynamics modelling are some examples where I anticipate that inducing program structure in the models can help increase robustness and reduce the amount of data needed for learning. Outside of robotics, I plan to apply my techniques on healthcare, computational biology, and finance applications where it is beneficial to have non-opaque and interpretable models. Image/Scene generation is another domain where compositionality plays a key role, and I believe neurosymbolic approaches can help recover the underlying modular representations for these problems.

There are several exciting research directions that use machine learning to improve computer systems and computer programming. The idea of incorporating program structure into neural models is not only important for AI systems, but also for big-data based code assistant systems; neurosymbolic approaches can learn compositional models from code databases that can systematically generalize to coding problems in both the same domain and other similar domains where the data might be limited. This will close the loop i.e. we can use programming to help machine learning to help programming itself.

Architectures: I want to explore other kinds of neurosymbolic architectures for the models. So far, my architectures involve parametric programs with loops, conditionals, and simple list operations; in the future, I want to explore architectures that include programs that can manipulate complex data-structures. Data-structures are instrumental in efficiently organizing the internal state of a model which in turn can lead to better generalization. The fundamental challenge, here, would be to find an over-parameterized representation that can meaningfully represent these complex data-structures and is also amenable to gradient-based approaches.

Another direction I am interested in pursuing is enabling architectures with both neural components and program components, where the programs capture the structure and logical part of the model and the neural networks capture the complexity of the real-world. For example, in the control problem in Figure 1, the input state of the car includes its geometrical positions and velocities. However, to learn a control policy that takes more complex inputs such as camera images or LIDAR images, we want a neural network to handle the perception part, but still have a program component to represent the high-level logic.

Algorithms: As we try out more applications and architectures, we will need more techniques to reason jointly about the symbolic and the continuous components. An interesting direction in this space is to be able to move dynamically the boundary between what is handled by symbolic programs and what is handled by neural networks. My ultimate goal for neurosymbolic learning is to have an analog of SGD (stochastic gradient descent), ADAM, or other optimization algorithms, i.e. we need one (or a small set of) general purpose algorithm(s) that can handle various neurosymbolic architectures/applications.

Another benefit of learning programmatic models is that we can leverage the fact that formal verification methods are more mature at handling structured programs rather than neural networks. In this respect, I want to develop algorithms to formally verify that the neurosymbolic models learned for an intelligent system are safe, robust, and fair.

References

[1] Thais Campos, Jeevana Priya Inala, Armando Solar-Lezama, and Hadas Kress-Gazit. Task-based design of ad-hoc modular manipulators. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6058–6064. IEEE, 2019.

[2] Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. Inversecsg: Automatic conversion of 3d models to csg trees. *ACM Transactions on Graphics (TOG)*, 37(6):1–16, 2018.

[3] Jeevana Priya Inala, Osbert Bastani, Zenna Tavares, and Armando Solar-Lezama. Synthesizing programmatic policies that inductively generalize. In *International Conference on Learning Representations*, 2020.

[4] Jeevana Priya Inala, Yecheng Jason Ma, Osbert Bastani, Xin Zhang, and Armando Solar-Lezama. Safe human-interactive control modulo fault. *Under submission*, 2020.

[5] Jeevana Priya Inala and Rishabh Singh. Webrelate: integrating web data with spreadsheets using examples. *Proceedings of the ACM on Programming Languages*, 2(POPL):1–28, 2017.

[6] Jeevana Priya Inala, Rohit Singh, and Armando Solar-Lezama. Synthesis of domain specific cnf encoders for bit-vector solvers. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 302–320. Springer, 2016.

[7] Jeevana Priya Inala, Yichen Yang, James Paulos, Yewen Pu, Osbert Bastani, Vijay Kumar, Martin Rinard, and Armando Solar-Lezama. Neurosymbolic transformers for multi-agent communication. *Advances in Neural Information Processing Systems*, 33, 2020.