Tell me and I forget, teach me and I may remember, *involve* me and I *learn*. —Benjamin Franklin

My own learning, teaching, mentoring experience has strengthened my belief in this philosophy and developed it into my principle of teaching and mentoring:

I *involve* students in *learning*, empowering them to build *their own* architecture of knowledge.

When I teach a class or mentor research students, this principle naturally applies as I involve students in:

designing curriculum. Whenever teaching a class, I carefully study (e.g., by survey or personal communication) the background of the students and their motivations in taking the class, and then tailor my course materials towards their interests. This usually makes the students motivated and engaged in learning. For example, when teaching Convex Optimization at TTIC and UChicago, I tailored the course materials towards the students' interest in machine learning,[1] covering stochastic gradient methods which are essential tools in training modern models (e.g., neural nets) but not included in most introductory optimization courses. In addition, I sent out weekly surveys to collect feedback from students, according to which I adjusted the pace of lectures and the amount of homework. Despite being new to the role of primary instructor, I received positive feedback from students at the end of the semester: one of the students enthusiastically rated my course as "the best I have ever taken".

(re)inventing ideas. While teaching a technical idea, I like to walk students through a series of questions and hints, inspiring them to come up with the idea by themselves: this often leads to a deep understanding and high excitement. Afterwards, I ask them to explain the idea to me, through which they may further deepen their understanding as well as improve their expressiveness. For example, when teaching Newton's method in my Convex Optimization course, I started from a quadratic function, asked students to figure out the direction of steepest descent under the quadratic norm induced by Hessian, guided them to generalize what they found to general convex functions, and inspired them to reinvent the key idea of "minimizing a local quadratic approximation to a function".

(re)implementing ideas. Implementing a technical idea is often an effective way of "debugging" one's understanding of it. When teaching a class, I like to include programming questions in the assignments, designing a series of test cases for each programming question and asking the students to interpret the output of their programs on those cases. When assisting Prof. Eisner (my PhD advisor) in teaching the course Natural Language Processing (NLP), I took initiative to build an autograder that is able to automatically evaluate the students' code (e.g., correctness, efficiency) and generate textual feedback. This significantly increased the working efficiency of us and future teaching assistants, enabling the size of the class to be enlarged such that a larger number of interested students can enroll in it.

By adopting these practices, I achieved successful teaching experience as the teaching assistant of Prof. Eisner's NLP, as a guest lecturer of Prof. Yarowsky's Information Retrieval and Web Agents, as a guest speaker for the JHU Summer School on Human Language Technology, and as the primary instructor of my Convex Optimization course. Moreover, I successfully mentored several undergraduate and Masters research students:[2] under my mentorship, they developed good research taste, acquired useful technical skills, and ended up publishing papers with me at prestiguous conferences such as ICML, NeurIPS, and EMNLP. Most of them received PhD offers from top universities such as Johns Hopkins University, University of Chicago, University of Maryland College Park, and Toyota Technological Institute at Chicago.

Combining my teaching, mentoring and research experience, I will be able to enrich the curriculum of the department that I'll join with my unique perspectives. Now I'll introduce the courses that I can offer.

[1] I am aware of the students' interest because I received my Masters from UChicago and I consulted previous course instructors.

[2] I was the primary research advisor and thesis advisor for two Masters students at the University of Chicago.

Core Computer Science and Statistics Courses

I am capable of teaching a wide range of core computer science and statistics courses, including but not limited to Data Structures, Algorithms, Artificial Intelligence, Machine Learning, Probability and Statistics, and Linear Algebra. Thanks to my research background, I could provide unique perspectives on teaching these courses. For example, for Data Structures and Algorithms, I could expose students to the machine learning problems in which classical data structures and algorithms play essential roles (e.g., nearest neighbor search via locality sensitive hashing, structure inference with dynamic programming).

Machine Learning for Time Series

Time series data is ubiquitous in real-world domains such as healthcare and finance. Such data includes

- *regular time series* (i.e., sequences with equal time intervals) such as daily and weekly stock prices.
- *irregular time series* (i.e., time-stamped sequences with stochastic time intervals) such as electronic health records, streams of Tweets, and logs of online browsing and purchases.

My research experience in event sequence modeling has prepared me to teach a course that covers both kinds of time series and a wide range of methods to handle them. I can introduce general principles for designing models and algorithms for time series, and discuss how specific methods are developed under the guidance of these principles. In addition, I can expose students to open research questions.

Natural Language Processing

My teaching and research experience has prepared me to offer a rich course on natural language processing, covering linguistics-motivated models (e.g., grammars, parsers, part-of-speech taggers), neural methods (e.g., recurrent neural networks, Transformer, BERT, GPT), and their combinations (e.g., neural parsers, neural module networks, ordered neurons). I will also cover interdisciplinary applications of natural language technologies, where NLP models are connected with models of other disciplines to handle data from multiple modalities. Such applications include event prediction and language-instructed robotics.

Language Modeling

In response to the recent success of large language models (LLMs), I am enthusiastic about introducing an innovative course in language modeling. This course aims to provide students with a comprehensive understanding of the field's most fundamental concepts and most advanced techniques. The curriculum will span a wide spectrum of topics, starting with n-gram models and statistical language modeling, gradually progressing to more sophisticated models including Transformers. Importantly, the course will have a dedicated focus on transformative advancements like Generative Pretrained Transformers (GPTs) and the development of sophisticated language agents. Through hands-on projects, students will have the opportunity to gain proficiency in implementing these techniques and applying them in practical scenarios.

Neuro-Symbolic Machine Learning

Traditional artificial intelligence courses mostly focus on symbolic methods for planning and reasoning, while modern machine learning courses mostly focus on pure data-driven approaches such as neural networks. I am able to offer a course at the intersection, exposing the next generation of scientists and engineers to methods of both categories and, more importantly, ways of combining them. This course will start with graphical models and algorithms for searching their structures, followed by logic languages (e.g., propositional logic and first-order logic) and ways of using these languages to ground the random variables and edges of a graphical model to symbolic meanings (e.g., objects and relations). Then it'll discuss some signature families of grounded graphical models (e.g., Markov logic networks and Bayesian logic programs) as well as algorithms for learning their parameters and structures and doing inference with them. Then it'll cover how to ground neurons in a neural network to symbolic meanings and how to train and do inference with this grounded network (e.g., neural module network).