What grade would you like to receive in this course? What grade do you honestly believe you deserve? Justify your answers.

- S
- S
- I have completed all assignments, reviewed all lecture videos, and practiced the recommended exercises and submitted these to the TA on time.

What have you learned in this course? Explain in more abstract, high-level terms (don't just list technical details from lecture slides). Has your view of AI changed from what it was before taking this class (in what way)?

The core material in this class shows how we can build an arbitrarily sized neural network, and why it works:

- First, we can think of neural networks as a series expansion in a mathematical sense, they are able to approximate any arbitrary function, given enough layers, i.e., Cybenko and the Universal Approximation Theorem.
- Learning can be mathematically defined as a process by which we optimize some sort of loss function. The idea is to get the loss as close to 0 as possible. Backpropagation, due to its efficiency in computing the gradient of the loss with respect to the weights of any arbitrarily sized network for each single input-output instance, has made it the favored algorithm for this process.
- A powerful method for implementing neural networks of arbitrary complexity is achievable if we abstract neural networks as computation graphs. If we take care to abstract the atomic operations, i.e., the nodes, writing the outputs as flowing through a graph in a forward pass, and storing the values for each operation in a systematic way, we can compute the backwards pass after calculating the loss function in such a way that the differentiation at each node can be automated.
- Being able to automatically differentiate each node in the backwards pass solves the otherwise intractable problem with regards to the Chain Rule, i.e., the combinatorial explosion of partial derivatives as multiple paths emerge in a growing computation graph, and allows us to dynamically produce computation graphs of arbitrary complexity.
- We can accomplish the above by taking advantage of object-oriented programming in Python. If we efficiently make use of classes and objects, we can automate the back-propagation process. Doing so allows us to deal with higher dimensional vector spaces by treating non-linear activation functions (sigmoid, tanh, cross-entropy) in the same way as our atomic operations (addition, multiplication, etc.) in code, i.e., if we treat them as just instance methods of a tensor class, whenever we call them on some tensor in order to return a new tensor, we can store pointers to their creators and creation operations in memory so that their derivatives with respect to the loss can be automatically computed in the backwards pass.

My view of AI has become more mature. Although I was interested in this topic before taking the class, I lacked the mathematically maturity to understand important concepts underlying modern deep learning approaches. While many of the more philosophical ideas presented were familiar to me, I'm grateful that this course did not handwave the mathematical foundations, while providing a broad historical overview of the field. It has

given me a better appreciation for the multidisciplinary nature of artificial intelligence, and has provided a good survey of the necessary material for further study.

What would you like to learn in an introductory AI course (e.g. if you take this course in a future semester)? Are you interested in more pragmatic coding with current industry libraries or in understanding the fundamental principles and theory? Do you prefer to zoom out and focus less on the mathematical/programmatic details of AI systems (instead discussing recent trends on a higher level of abstraction), or do you prefer to zoom in and focus more closely on mathematics and coding of foundational algorithms?

- I would like to have at least one assignment where we make use of PyTorch or TensorFlow because they are widely utilized frameworks. Building some familiarity with one would be useful for future projects.
- I would like to have at least one applied version of the assignments. For example, having students design their own learning problem (either in groups or individually) and submitting it in lieu of a final examination. Our assignments were defined problems. This was advantageous because the application was clear and the datasets were pre-processed, allowing students to focus on the evaluation of their models. However, rephrasing a real-life problem into an implementable learning problem is a practical skill I would have liked to develop, namely, finding suitable data on my own and cleaning/processing it so that it could be used as training data.
- Continue to Introduce programming assignments with detailed explanations, where almost every part of the code is explained in detail. The detail given for the template code in Assignment 2 was enough that a studious novice could complete the assignment. Consider uploading videos for coding explanations even if the classes are presented live in the future. Having access to high quality videos with coding explanations was extremely useful for note-taking, and the teacher/TA will not have to waste time re-explaining things covered in the lecture. This is my personal experience, but I needed to listen to lectures regarding coding several times over while taking notes to fully absorb the content. This wouldn’t have been possible in a live class.