Interactive Assignments for Teaching Structured Neural NLP

David Gaddy, Daniel Fried, Nikita Kitaev, Mitchell Stern, Rodolfo Corona, John DeNero and **Dan Klein**
University of California, Berkeley
{dgaddy,denero,klein}@berkeley.edu

Abstract

We present a set of assignments for a graduate-level NLP course. Assignments are designed to be interactive, easily gradable, and to give students hands-on experience with several key types of structure (sequences, tags, parse trees, and logical forms), modern neural architectures (LSTMs and Transformers), inference algorithms (dynamic programs and approximate search) and training methods (full and weak supervision). We designed assignments to build incrementally both within each assignment and across assignments, with the goal of enabling students to undertake graduate-level research in NLP by the end of the course.

1 Overview

Our course contains five implementation projects focusing on neural methods for structured prediction tasks in NLP. Over a range of tasks from language modeling to machine translation to syntactic and semantic parsing, the projects cover methods such as LSTMs and Transformers, dynamic programming, beam search, and weak supervision (learning from denotations). Our aim was to let students incrementally and interactively develop models and see their effectiveness on real NLP datasets. Section 2 describes the tasks and objectives of the projects in more detail. Links to assignments are available at `https://sites.google.com/view/nlp-assignments`.

1.1 Target Audience

Our course is designed for early-stage graduate students in computer science. We expect students to have prior experience in machine learning, including deep learning. Many of our students will go on to conduct research in NLP or related machine learning disciplines. Some advanced undergraduates may also join the course if they have sufficient background and an interest in NLP research.

1.2 Design Strategy

All projects are implemented as interactive Python notebooks designed for use on Google's Colab infrastructure.[1] This setup allows students to use GPUs for free and with minimal setup. The notebooks consist of instructions interleaved with code blocks for students to fill in. We provide scaffolding code with less critical components like data loading already filled in, so that students can focus on the learning objectives for the projects. Students implement neural network components using the PyTorch framework (Paszke et al., 2019).

Each project is broken down into a series of modules that can be verified for correctness before moving on. For example, when implementing a neural machine translation system, the students first implement and verify a basic sequence-to-sequence model, then attention, and finally beam search. This setup allows students to debug each component individually and allows instructors to give partial credit for each module. The modules are designed to validate student code without waiting for long training runs, with a total model training time less than one hour per project.

Our projects are graded primarily with scripted autograders hosted on Gradescope,[2] allowing a class of hundreds of students to be administered by a small course staff. Grades are generally based on accuracy on a held-out test set, where students are given inputs for this set and submit their model's predictions to the grader. While students cannot see their results on the held-out set until after the due date, the assignments include specific targets for validation set accuracies that can be used by students to verify the correctness of their solutions.

Each project concludes with an open-ended section where the students experiment with modifications or ablations to the models implemented and

[1] `colab.research.google.com`
[2] `www.gradescope.com`

submit a 1-page report describing the motivation behind their contribution and an analysis of their results. This section gives students more of a chance to explore their own ideas, and can also help distinguish students who are putting in extra effort on the projects.

2 Assignments

2.1 Project 0: Intro to PyTorch Mini-Project

This project serves primarily as an introduction to the project infrastructure and to the PyTorch framework. Students implement a classifier to predict the most common part of speech tag for a word type from its characters. They first implement a simple model based on pooling character embeddings, then a slightly more complex model with character n-gram representations. This project provides much more detailed instructions than later projects to help students who are less familiar with deep learning implementation, walking them through each step of the training and modeling code.

2.2 Project 1: Language Modeling

In this project, students implement a series of language models of increasing complexity. First they implement a basic n-gram model, then add backoff and Kneser-Ney smoothing (Ney et al., 1994). Next, they implement a feed-forward neural n-gram model, and an LSTM language model (Hochreiter and Schmidhuber, 1997). The last section of this project is an open-ended exploration where students can try any method to further improve results, either from a list of ideas we provided or an idea of their own.

2.3 Project 2: Neural Machine Translation

Students incrementally implement a neural machine translation model to translate from German to English on the Multi30K dataset (Elliott et al., 2016). This dataset is simpler than standard translation benchmarks and affords training and evaluating an effective model in a matter of minutes rather than days, allowing students to interactively develop and debug models. Students first implement a baseline LSTM-based sequence-to-sequence model (Sutskever et al., 2014) without attention, view the model's predictions, and evaluate performance using greedy decoding. Then, students incrementally add an attention mechanism (Bahdanau et al., 2015) and beam search decoding. Finally, students visualize the model's attention distributions.

2.4 Project 3: Parsing and Transformers

This project covers constituency parsing and the Transformer neural network architecture (Vaswani et al., 2017). Students first implement a Transformer encoder and validate it using a part-of-speech tagging task on the English Penn Treebank (Marcus et al., 1993). Then, students incrementally build a Transformer-based parser by first constructing a model that makes constituency and labeling decisions for each span in a sentence, then implementing CKY decoding (Cocke, 1970; Kasami, 1966; Younger, 1967) to ensure the resulting output is a tree. The resulting model is a small version of the parser of Kitaev and Klein (2018), and achieves reasonable performance on the English Penn Treebank in under half an hour.

2.5 Project 4: Semantic Parsing

In this project, students implement a neural semantic parser for the GeoQA geographical question answering dataset of Krishnamurthy and Kollar (2013). To familiarize themselves with the syntax and semantics of the dataset, students first implement a simple execution method which evaluates a logical form on a database to produce an answer. To produce logical forms from questions, students then implement a sequence-to-sequence architecture with a constrained decoder and a copy mechanism (Jia and Liang, 2016). Students verify their model by training first in a supervised setting with known logical forms, then finally train it only from question-answer pairs by searching over latent logical forms.

3 Findings in Initial Course Offering

An initial iteration of the course was taught to 60 students, and an offering for over 100 students is in progress. In the first iteration of the course, 81% of students completed the course and submitted all five projects. From a mid-semester survey, students reported taking 19.88 hours in average to complete Project 1. Most students were able to successfully complete the majority of each assignment, but there remained some point spread to distinguish performance for grading (mean 94%, standard deviation 12%). Due to the use of autograding, instructor grading effort totaled less than several hours per project. At the end of last year's course, one student submitted an extension of their exploration work on one project to EMNLP and presented at the conference.

References

Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015*.

John Cocke. 1970. Programming languages and their compilers: Preliminary notes.

Desmond Elliott, Stella Frank, Khalil Sima'an, and Lucia Specia. 2016. Multi30K: Multilingual English-German image descriptions. In *Proceedings of the 5th Workshop on Vision and Language*, pages 70–74, Berlin, Germany. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Berlin, Germany. Association for Computational Linguistics.

Tadao Kasami. 1966. An efficient recognition and syntax-analysis algorithm for context-free languages. *Coordinated Science Laboratory Report no. R-257*.

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686.

Jayant Krishnamurthy and Thomas Kollar. 2013. Jointly learning to parse and perceive: Connecting natural language to the physical world. *Transactions of the Association for Computational Linguistics*, 1:193–206.

Mitch Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

Hermann Ney, Ute Essen, and Reinhard Kneser. 1994. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language*, 8(1):1–38.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32:8026–8037.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 27:3104–3112.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*, 30:5998–6008.

Daniel H. Younger. 1967. Recognition and parsing of context-free languages in time n3. *Information and control*, 10(2):189–208.