5th International Conference on Higher Education Advances (HEAd'19)
Universitat Politècnica de València, València, 2019
DOI: http://dx.doi.org/10.4995/HEAd19.2019.9177

Deep Teaching: Materials for Teaching Machine and Deep Learning

Christian Herta, Benjamin Voigt, Patrick Baumann, Klaus Strohmenger, Christoph Jansen, Oliver Fischer, Gefei Zhang, Peter Hufnagel
Faculty 4, HTW Berlin - University of Applied Science, Germany

Abstract

Machine learning (ML) is considered to be hard because it is relatively complicated in comparison to other topics of computer science. The reason is that machine learning is based heavily on mathematics and abstract concepts. This results in an entry barrier for students: Most students want to avoid such difficult topics in elective courses or self-study.

In the project Deep.Teaching we address these issues: We motivate by selected applications and support courses as well as self-study by giving practical exercises for different topics in machine learning. The teaching material, provided as jupyter notebooks, consists of theoretical and programming sections.

For didactical reasons, we designed programming exercises such that the students have to deeply understand the concepts and principles before they can start to implement a solution. We provide all necessary boilerplate code such that the students can primarily focus on the educational objectives of the exercises. We used different ways to give feedback for self-study: obscured solutions for mathematical results, software tests with assert statements, and graphical illustrations of sample solutions. All of the material is published under a permissive license. Developing jupyter notebooks collaboratively for educational purposes poses some problems. We address these issues and provide solutions/best practices.

Keywords: *Machine learning; education; jupyter notebook; programming exercise; collaborative development.*

1. Introduction

Machine learning for students of computer science is relatively hard. It requires a strong background in mathematics even more than other areas of computer science. The students have to learn a lot of theoretical concepts and principles before they can deeply understand a specific algorithm or an application based on such.

To our experience, students avoid courses and self-study of difficult topics because they are discouraged by the reasons given above. Machine learning courses are typically not obligatory. However, their necessity is not doubtable. In the last couple of years machine learning algorithms, especially artificial neural networks, improved the results in some fundamental problems of computer science, e.g., image classification (He, 2016), speech recognition (Oord, 2016) or natural language processing (Bahdanau, 2014). Therefore ways have to be found to motivate students for machine learning.

In the project Deep.Teaching we describe three applications which are heavily based on machine learning. For each application, we also give competitions in which the students can participate in teams. Based on our experience, working on such projects increases the initial motivation and reduces difficulties in understanding theoretical concepts. It also deepens the students' knowledge substantially.

Typically a computer science course at our university consists of two elements: lectures and lab sessions. For most machine learning lectures we found good books, tutorials, and other teaching materials mainly focusing on the theory. However, for the lab sessions, there is a need for structured and semantically related exercises. Such exercises should address the relevant topics taught in the lectures respectively *flipped classroom* (Bergmann et al. 2012). In the project Deep.Teaching we developed various such exercises based on jupyter notebooks (https://jupyter.org/) to accommodate the needs. The notebooks are focusing on particular topics of machine learning, and we published all materials under a permissive license on the project website (https://deep-teaching.org).

Unfortunately, these notebooks also have disadvantages if a version control system is used for collaborative development. Jupyter notebooks store many metadata not immediately visible to the user but to the version control system. This property leads to unnecessary merge conflicts and makes development inconvenient.

Another challenge is to give the students the possibility to compare their exercise solution to a sample solution. Usually, all content is visible inside a notebook, so any provided solution would be visible as well. Therefore it is necessary to find a way to obfuscate or disguise the correct answer. This paper explains technics we used to handle that obstacle and describes best practice for collaborative work and sharing jupyter notebooks as teaching materials.

2. Related Work

Different authors described how they used jupyter notebooks for teaching and elementary procedures for working with notebooks in an educational context, see e.g., O'Hara et al. (2015) or Granado et al. (2018).

Lately, some authors and publishing companies also released books online, freely available as interactive *jupyter* notebooks, in addition to the printed version. Examples are the Python Data Science Handbook (VanderPlas, 2016) or Bayesian Methods for Hackers (Davidson-Pilon, 2015).

The project *nbgrader* (https://nbgrader.readthedocs.io/en/stable/) addresses the issue workflow, i.e., how to maintain separate instructor/student versions of a notebook. Its focus lies on how to (automatically) grade the students' solutions and seems to work best in combination with *jupyter* hub, a thin client/server solution, e.g., for classrooms or research labs (https://jupyter.org/hub).

3. Application Scenarios and Jupyter Notebook Exercises

3.1. Application Scenarios

An application scenario should transfer theoretical knowledge in a practical setting. An exciting environment of the scenario is highly essential. In order for the scenario to generate a strong motivation among students, the scenario should, e.g., open a professional perspective or be socially relevant. Functional requirements of teaching are also part of a good scenario choice. So it must be possible to depict the content of the course in the practical setting. Ideally, during the course, small exercises build a solution to a complex problem in the domain. Our scenarios are:

Medical Image Classification focuses primarily on detecting tumors in digitized histopathological images. Main teaching content is the knowledge of convolutional neural networks but also contains portions of fundamental machine learning algorithms.

Robotic Autonomous Driving deals with content required to control a robotic vehicle. Due to the variety of problems, different machine learning paradigms and algorithms are applied in this context. To further motivate students, we developed a framework to control a racecar in a simulated environment and real world (https://gitlab.com/NeuroRace).

Text Information Extraction/Natural Language Processing is a scenario used by many on a daily base, e.g., chatbots or search engines. We use this scenario to provide examples for sequence learning and corresponding algorithms.

A more detailed description of the scenarios is available online on the project website. With the selected scenarios, we cover a wide range of different problem areas and algorithms for machine learning.

In courses at our university and at summer schools the students work on the application scenarios and similar competitions. By working on software solutions. they feel the need to concern themselves with the necessary machine learning fundamentals. The positive feedback from students confirmed us that such scenarios are very motivating.

3.2. Jupyter Notebook Exercises

As a tool for the exercises, we use *jupyter* notebook, which is an interactive environment mainly developed for data science and machine learning (Shen, 2014). *Jupyter* notebooks are documents structured in cells for source code, visualization, mathematical equations, and text.

Jupyter notebook can be used as an environment for both theoretical concepts and exercises. The description of the exercises can be given within the same environment where the students have to implement their solutions. To prepare the students many of our notebooks also contain pen & paper exercises in addition to the programming parts. In most cases, we provide a short review of the necessary theory at the beginning, together with links to literature for further reading. Also within the notebook, code to generate plots and diagrams to visualize the results of the implementation can be predefined. The students can interactively manipulate an input of a code snippet or a mathematical procedure, which results in different behavior of the function and produce another output. Students immediately see results of their changes visually. The effects of such changes become vivid and less abstract.

3.3. Didactic Concepts and Structure of Notebooks

For developing our exercises we use the following (didactic) guidelines. Examples are more illustrative than abstract descriptions. The examples should be as simple as possible that the students can focus on the teaching objective, i.e., we follow the *didactic reduction* principle (Wüest, 2018). We provide all necessary additional helper functions, e.g., for data loading and visualization of the results. This way the students can focus on the learning objectives without the need to implement disturbing *boilerplate code*.

We design the notebooks such that each one supports the learning of one particular subject. This does not exclude that several notebooks chained can lead to the solution of a more complex task. In general, we prefer small notebooks focusing on only one concept. Whenever possible during development of the notebooks we divide larger notebooks in smaller ones. We also tried to avoid strong dependencies between notebooks, which allows

using notebooks in different courses or application scenarios. All notebooks follow the same design and structure. Our blueprint structure is:

Introduction defines the learning objective and describes the structure of the exercise.

Prerequisite provides all necessary information or sources a student needs to solve the exercise. We also provide all required python-modules and data in that section.

Exercise contains instructions for the programming assignments and provides an overview of helper functions used in the exercise. It is also possible to recap certain theory aspects to clarify concepts of the learning objective.

Outlook summarizes the learning content and gives further information to related topics or exercises.

Each notebook ends with a summary of the literature used and a license under which the notebook is published. Besides, each notebook contains a table of content for navigation and optionally an acknowledgment section.

3.4. Feedback and Tests

Our notebooks contain attached images for comparison, e.g., the progress of training a model or the final decision boundary in a classification task. This way students can check visually if their results match with the sample solution images. This also helps teachers when sighting programming assignments. They do not need to stumble through the complete code. If the visualization of a result seems accurate, the corresponding implementation is likely to be correct. Complementary to the visual feedback, we also provide software tests. The tests check if the implementations behave correctly, by comparing their output with the solution. However, the nature of some machine learning tasks is that not all algorithms behave completely deterministic. In these cases, directly testing the output might not be possible, which shows the importance of providing sample solution images of the visualized results.

4. Lessons Learned / Best Practices

This section describes the lessons we learned and the best practices for collaboratively developing and publishing *jupyter* notebooks for educational purposes.

4.1. Workflow for Generating Notebooks

When developing exercise notebooks we create two cells for the solution of a single exercise: A (semi) blank cell, where the students shall implement a function or fill in missing code and a cell containing the sample solution. The student version of the notebooks should not contain the sample solution cells. Manually deleting and maintaining

two different notebooks is error prone and time-consuming. A way to automatically delete the solutions and to generate the student version is needed. For this purpose we utilized the slide type information of the *slideshow feature*, provided by the RISE extension (https://github.com/damianavila/RISE), to mark *solution-cells* as *skip cells*. Moreover, the cells containing the sample solutions can be removed automatically by a script we developed for this use case (also open sourced).

We used two repositories for version control: One for development and one for publishing the notebooks for the students. Openly published notebooks do not contain solutions. Each notebook can be accessed via a web link, so it is easy to reference them from course sites, e.g,. moodle. Teachers can request access to a private repository, which contains the same exercises including the sample solutions.

4.2. Version Control

For the collaborative development of notebooks we used the version control system git (https://git-scm.com). A version control system is typically used for collaborative development and versioning of source code. For collaborative development, one goal was to keep the infrastructure effort as low a possible. So we used *gitlab* (https://gitlab.com/) rather than a self-hosted git infrastructure.

The version control system keeps track of the changes in a text file, which might come from different contributors and in most cases manages to solve conflicts automatically. Jupyter notebooks are stored in JSON-Format (Bray, 2017), which is harder to parse for the control systems (as well as for humans) than plain text.

Another problem is that *jupyter* notebook (the environment) stores meta information together with the notebook content, which results in differences on the text level, e.g., date and time of last execution. This typically results in a merge conflict, which means that the version control system cannot automatically bring the different versions from different contributors together, even if a collaborator has only executed the notebook without modifying anything. To tackle this problem we developed a script which is executed before the updated notebooks are uploaded and removes this unnecessary data to avoid such conflicts.

4.3. Software Requirements

For each notebook we provide a file with software requirements, enabling the user to install the needed packages in a particular version easily. This is crucial as the behavior or naming of external libraries' functions might change in newer versions.

Nevertheless, dependencies to external software libraries should be minimized. Typically modern machine learning libraries underlie frequent API changes, which would either

require to update notebooks permanently to keep them up-to-date or rely on the named file which specifies an older but tested version. However, teaching outdated library usage should be avoided when possible. So we prefer to write notebooks which depend just on fundamental and stable libraries such as numpy (http://numpy.org). Numpy is a basic linear algebra library for working with multi-dimensional arrays similar to Matlab (https://mathworks.com/). That is also in accordance with our teaching goals: The students should learn principles and fundamentals and not special API-calls or software libraries.

For teaching neural networks we are developing minimalistic deep learning libraries from scratch based on numpy only. Corresponding notebooks with exercises show how a modern deep learning library works in principle under the hood. The main functionality of such libraries is automatic gradient calculations (Goodfellow et. al., 2016).

4.4. Numeric Value Tests

The result of many mathematical and programming exercises are numerical values. To provide a software test as direct feedback for the students without spoiling the solution, we use hash functions to obscure the real values. A hash function is a one-way function: it is easy to compute a hash value from the input value, but it is prohibitively expensive to get the input value from the hash value. A typical solution value is a floating point number. So it is essential to take into account that solutions can be given with different numerical precisions. To circumvent that, the solution test rounds the values to the same precision before applying the hash function. An example as code snippet is found in *exercise: evaluation metrics* (https://dev.deep-teaching.org/notebooks/machine-learning-fundamentals/exercise-evaluation-metrics).

5. Discussion and Outlook

This paper aims to present the results of the project Deep.Teaching. Here we discuss possible further developments, improvements, and studies for future work.

We plan to extend and review the notebooks thoroughly in the next year. For this, we would appreciate if we get feedback from teachers of other universities, e.g., by issue tracking (https://gitlab.com/deep.TEACHING/educational-materials/issues).

We want to embed the notebooks in example courses on our website. Each example course should correspond to a web page where the course content is described, and pointers are given to literature for self-study.

At the moment students get just qualitative feedback if they succeeded in solving exercises. If a test fails they know that their solution cannot be entirely correct. For university courses, it would be helpful if the students are graded automatically, i.e., giving them points for

solving exercises. The *nbgrader* project (https://nbgrader.readthedocs.io) addresses this issue. *Nbgrader* requires a special directory structure which does not fit to our current (collaborative) development workflow. However, for teachers, it should be easy to reorganize the notebooks for a course such that they can use *nbgrader* without much effort.

By using *jupyter* notebook exercises in our courses we conclude that *jupyter* notebook is an adequate environment for teaching. We also experienced that abstract concepts can be learned interactively by programming much more clearly and insightful as with mathematical formulas and text alone. We leave it to future work to study this hypothesis quantitatively.

Acknowledgment

The project Deep.Teaching is funded by the German National Ministry of Education and Research (BMBF), project number 01IS17056.

References

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bergmann, J., Sams, A. (2012). Flip your classroom: Reach every student in every class every day. *International society for technology in education.*

Bray, T. (2017). The javascript object notation (json) data interchange format (No. RFC 8259).

Davidson-Pilon, C. (2015). Bayesian methods for hackers: probabilistic programming and Bayesian inference. Addison-Wesley Professional.

Díaz García, E., & Cabrera Granado, E. (2018). Guide to Jupyter Notebooks for educational purposes, *Technical report, Universidad Complutense de Madrid*. Retrieved from http://eprints.ucm.es/48305/1/ManualJupyterIngles.pdf

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1). Cambridge: MIT press.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

Oord, A. V. D., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... & Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

O'Hara, K. J., Blank, D., & Marshall, J. (2015). Computational notebooks for AI education. *In Proceedings of the Twenty-Eighth International Florida Artificial Intelligence Research Society Conference*

Shen, H. (2014). Interactive notebooks: Sharing the code. Nature News, 515(7525), 151.

VanderPlas, J. (2016). Python data science handbook: essential tools for working with data. " O'Reilly Media, Inc.".

Wüest, Y., Zellweger, F., (2018). Strategies to reduce learning content, Chapter 3. In Competence Oriented Teaching and Learning. Ed.: H. Bachmann, HEP-Verlag, ISBN-13: 978-3035512373