Part I

Appendix: Tooling

Appendix A

DataDeps.jl: Repeatable Data Setup for Replicable Data Science

This paper is currently under review for the Journal of Open Research Software.

Abstract

We present DataDeps.jl: a julia package for the reproducible handling of static datasets to enhance the repeatability of scripts used in the data and computational sciences. It is used to automate the data setup part of running software which accompanies a paper to replicate a result. This step is commonly done manually, which expends time and allows for confusion. This functionality is also useful for other packages which require data to function (e.g. a trained machine learning based model). DataDeps.jl simplifies extending research software by automatically managing the dependencies and makes it easier to run another author's code, thus enhancing the reproducibility of data science research.

A.1 Introduction

In the movement for reproducible sciences there have been two key requests upon authors: **1.** Make your research code public, **2.** Make your data public (Goodman et al. 2014). In practice this alone is not enough to ensure that results can be replicated. To get another author's code running on a your own computing environment is often non-trivial. One aspect of this is data setup: how to acquire the data, and how to connect it to the code.

DataDeps.jl simplifies the data setup step for software written in Julia (Bezanson et al. 2014). DataDeps.jl follows the unix philosophy of doing one job well. It allows the code to depend on data, and have that data automatically downloaded as required. It increases replicability of any scientific code that uses static data (e.g. benchmark datasets). It provides simple methods to orchestrate the data setup: making it easy to create software that works on a new system without any user effort. While it has been argued that the direct replicability of executing the author's code is a poor substitute for independent reproduction (Drummond 2009), we maintain that being able to run the original code is important for checking, for understanding, for extension, and for future comparisons.

Vandewalle, Kovacevic, and Vetterli (2009) distinguishes six degrees of replicability for scientific code. The two highest levels require that "The results can be easily reproduced by an independent researcher with at most 15 min of user effort". One can expend much of that time just on setting up the data. This involves reading the instructions, locating the download link, transferring it to the right location,

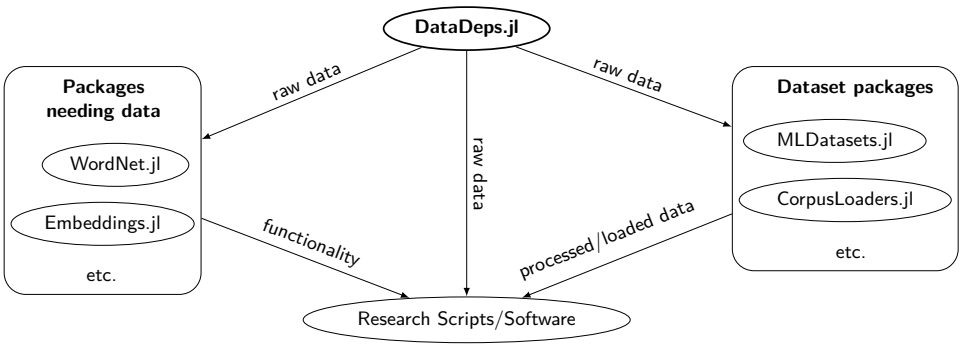

Figure A.1: The current package ecosystem depending on DataDeps.jl.

extracting an archive, and identifying how to inform the script as to where the data is located. These tasks are automatable and therefore should be automated, as per the practice "Let the computer do the work" (Wilson et al. 2014).

DataDeps.jl handles the data dependencies, while Pkg[1] and BinDeps.jl,[2] (etc.) handle the software dependencies. This makes automated testing possible, e.g., using services such as TravisCI[3] or AppVeyor.[4] Automated testing is already ubiquitous amongst julia users, but rarely for parts where data is involved. A particular advantage over manual data setup, is that automation allow scheduled tests for URL decay (Wren 2008). If the full deployment process can be automated, given resources, research can be fully and automatically replicated on a clean continuous integration environment.

A.1.1 Three common issues about research data

DataDeps.jl is designed around solving common issues researchers have with their file-based data. The three key problems that it is particularly intended to address are:

Storage location: Where do I put it? Should it be on the local disk (small) or the network file-store (slow)? If I move it, am I going to have to reconfigure things?

Redistribution: I don't own this data, am I allowed to redistribute it? How will I give credit, and ensure the users know who the original creator was?

Replication: How can I be sure that someone running my code has the same data? What if they download the wrong data, or extract it incorrectly? What if it gets corrupted or has been modified and I am unaware?

A.2 DataDeps.jl

A.2.1 Ecosystem

DataDeps.jl is part of a package ecosystem as shown in Figure A.1. It can be used directly by research software, to access the data they depend upon for e.g. evaluations. Packages such as MLDatasets.jl[5] provide more convenient accesses with suitable preprocessing for commonly used datasets. These packages currently use DataDeps.jl as a back-end. Research code also might use DataDeps.jl indirectly by making use of packages, such as WordNet.jl[6] which currently uses DataDeps.jl to ensure it has the data it depends on to function (see Appendix A.4.1); or Embeddings.jl which uses it to load pretrained machine-learning models. Packages and research code alike depend on data, and DataDeps.jl exists to fill that need.

[1] `https://github.com/JuliaLang/Pkg.jl`
[2] `https://github.com/JuliaLang/BinDeps.jl`
[3] `https://travis-ci.org/`
[4] `https://ci.appveyor.com/`
[5] `https://github.com/JuliaML/MLDatasets.jl`
[6] `https://github.com/JuliaText/WordNet.jl`

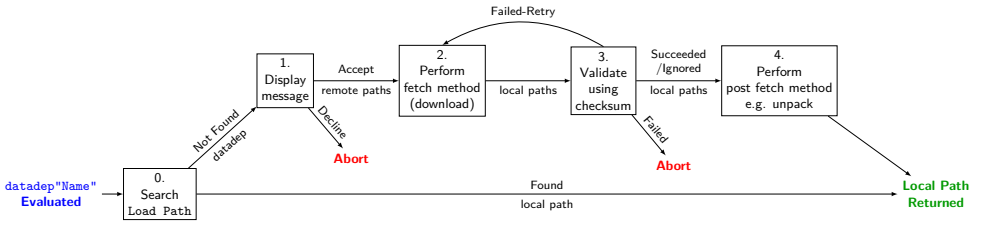

Figure A.2: The process that is executed when a data dependency is accessed by name.

A.2.2 Functionality

Once the dependency is declared, data can accessed by name using a datadep string written `datadep"Name"`. This can treated just like a filepath string, however it is actually a string macro. At compile time it is replaced with a block of code which performs the operation shown in Figure A.2. This operation always returns an absolute path string to the data, even that means the data must be download and placed at that path first.

DataDeps.jl solves the issues in Appendix A.1.1 as follows:

Storage location: A data dependency is referred to by name, which is resolved to a path on disk by searching a number of locations. The locations search is configurable.

Redistribution: DataDeps.jl downloads the package from its original source so it is not redistributed. A prompt is shown to the user before download, which can be set to display information such as the orignal author and any papers to cite etc.

Replication: when a dependency is declared, the creator specified the URL to fetch from and post fetch processing to be done (e.g. extraction). This removed the chance for human error. To ensure the data is exactly as it was originally checksum is used.

DataDeps.jl is primarily focused on public, static data. For researchers who are using private data, or collecting that data while developing the scripts, a manual option is provided; which only includes the **Storage Location** functionality. They can still refer to it using the `datadep"Name"`, but it will not be automatically downloaded. During publication the researcher can upload their data to an archival repository and update the registration.

A.2.3 Similar Tools

Package managers and build tools can be used to create adhoc solutions, but these solution will often be harder to use and fail to address one or more of the concerns in Appendix A.1.1. Data warehousing tools, and live data APIs; work well with continuous streams of data; but they are not suitable for simple static datasets that available as a collection of files.

Quilt[7] is a more similar tool. In contrast to DataDeps.jl, Quilt uses one centralised data-store, to which users upload the data, and they can then download and use the data as a software package. It does not directly attempt to handle any **Storage Location**, or **Redistribution** issues. Quilt does offer some advantages over DataDeps.jl: excellent convenience methods for some (currently only tabular) file formats, and also handling data versioning. At present DataDeps.jl does not handle versioning, being focused on static data.

[7] `https://github.com/quiltdata/quilt`

A.2.4 Quality Control

Using AppVeyor and Travis CI testing is automatically performed using the latest stable release of Julia, for the Linux, Windows, and Mac environments. The DataDeps.jl tests include unit tests of key components, as well as comprehensive system/integration tests of different configurations of data dependencies. These latter tests also form high quality examples to supplement the documentation for users to looking to see how to use the package. The user can trigger these tests to ensure everything is working on their local machine by the standard julia mechanism: running `Pkg.test(``DataDeps'')` respectively.

The primary mechanism for user feedback is via Github issues on the repository. Bugs and feature requests, even purely by the author, are tracked using the Github issues.

A.3 Availability

A.3.1 Operating system

DataDeps.jl is verified to work on Windows 7+, Linux, Mac OSX.

A.3.2 Programming language

Julia v0.6, and v0.7 (1.0 support forthcoming).

A.3.3 Dependencies

DataDeps.jl's dependencies are managed by the julia package manager. It depends on SHA.jl for the default generation and checking of checksums; on Reexport.jl to reexport SHA.jl's methods; and on HTTP.jl for determining filenames based on the HTTP header information.

List of contributors

- Lyndon White (The University of Western Australia) Primary Author
- Christof Stocker (Unaffiliated), Contributor, significant design discussions.
- Sebastin Santy (Birla Institute of Technology and Science), Google Summer of Code Student working on DataDepsGenerators.jl

A.3.4 Software location:

Name: oxinabox/DataDeps.jl
Persistent identifier: `https://github.com/oxinabox/DataDeps.jl/`
Licence: MIT
Date published: 28/11/2017
Documentation Language English
Programming Language Julia
Code repository GitHub

A.4 Reuse potential

DataDeps.jl exists only to be reused, it is a "backend" library. The cases in which is should be reused are well discussed above. It is of benefit to any application, research

tool, or scientific script that has a dependency on data for it's functioning or for generation of its result.

DataDeps.jl is extendible via the normal julia methods of subtyping, and composition. Additional kinds of `AbstractDataDep` can be created, for example to add an additional validation step, while still reusing the behaviour defined. Such new types can be created in their own packages, or contributed to the open source DataDeps.jl package.

Julia is a relatively new language with a rapidly growing ecosystem of packages. It is seeing a lot of up take in many fields of computation sciences, data science and other technical computing. By establishing tools like DataDeps.jl now, which support the easy reuse of code, we hope to promote greater resolvability of packages being created later. Thus in turn leading to more reproducible data and computational science in the future.

A.4.1 Case Studies

Research Paper: White et al. (2016) We criticize our own prior work here, so as to avoid casting aspersions on others. We consider it's limitations and how it would have been improved had it used DataDeps.jl. Two version of the script were provided[8] one with just the source code, and the other also including 3GB of data. It's license goes to pains to explain which files it covers and which it does not (the data), and to explain the ownership of the data. DataDeps.jl would avoid the need to include the data, and would make the ownership clear during setup. Further sharing the source code alone would have been enough, the data would have been downloaded when (and only if) it is required. The scripts themselves have relative paths hard-coded. If the data is moved (e.g. to a larger disk) they will break. Using DataDeps.jl to refer to the data by name would solve this.

Research Tool: WordNet.jl WordNet.jl is the Julia binding for the WordNet tool (Miller 1995). As of PR #8[9] it now uses DataDeps.jl. It depends on having the WordNet database. Previously, after installing the software using the package manager, the user had to manually download and set this up. The WordNet.jl author previously had concerns about handling the data. Including it would inflate the repository size, and result in the data being installed to an unreasonable location. They were also worried that redistributing would violate the copyright. The manual instructions for downloading and extracting the data included multiple points of possible confusion. The gzipped tarball must be correctly extracted. The user must know to pass in the *grand-parent* directory of the database files. Using DataDeps.jl all these issues have now been solved.

Acknowledgements

Thank particularly to Christof Stocker, the creator of MLDatasets.jl (and numerous other packages), in particular for his bug reports, feature requests and code reviews; and for the initial discussion leading to the creation of this tool.

Competing interests

The authors declare that they have no competing interests.

[8] Source code and data provided at `http://white.ucc.asn.au/publications/White2016BOWgen/`
[9] `https://github.com/JuliaText/WordNet.jl/pull/8`

A.5 Concluding Remarks

DataDeps.jl aims to help solve reproducibility issues in data driven research by automating the data setup step. It is hoped that by supporting good practices, with tools like DataDeps.jl, now for the still young Julia programming language better scientific code can be written in the future .

Appendix B

DataDepsGenerators.jl: making reusing data easy by automatically generating DataDeps.jl registration code

This paper is currently under review for the Journal of Open Source Software.

B.1 Summary

DataDepsGenerators.jl is a tool written to help users of the Julia programming language (Bezanson et al. 2014), to observe best practices when making use of published datasets. Using the metadata present in published datasets, it generates the code for the data dependency registration blocks required by DataDeps.jl (White et al. 2018). These registration blocks are effectively executable metadata, which can be resolved by DataDeps.jl to download the dataset. They include a message that is displayed to the user whenever the data set is automatically downloaded. This message should include provenance information on the dataset, so that downstream users know its original source and details on its processing.

DataDepsGenerators.jl attempts to use the metadata available for a dataset to capture and record:

- The dataset name.
- A URL for a website about the dataset.
- The names of the authors and maintainers
- The creation date, publication date, and the date of the most recent modification.
- The license that the dataset is released under.
- The formatted bibliographic details of any paper about or relating to the dataset.
- The formatted bibliographic details of how to cite the dataset itself.
- A list of URLs where the files making up the dataset can be downloaded.
- A corresponding list of file hashes, such as MD5 or SHA256, to validate the files after download.
- A description of the dataset.

Depending on the APIs supported by the repository some of this information may not be available. DataDepsGenerators.jl makes a best-effort attempt to acquire as much provenance information as possible. Where multiple APIs are supported, it makes use of all APIs possible, merging their responses to fill any gaps. It thus often produces higher quality and more comprehensive dataset metadata than is available from any one source.

DataDepsGenerators.jl leavages many different APIs to support a very large number of repositories. By current estimates tens of millions of datasets are supported, from hundreds of repositories. The APIs supported include:

- DataCite / CrossRef
  - This is valid for the majority of all dataset with a DOI.
- DataOne
  - This supports a number of data repositories used in the earth sciences.
- FigShare
  - A popular general purpose data repository.
- DataDryad
  - A data repository particularly popular with evolutionary biology and ecology.
- UCI ML repository
  - A data repository commonly used for small-medium machine learning benchmark datasets.
- GitHub
  - Most well known for hosting code; but is fairly regularly used to host versioned datasets.
- CKAN
  - This is the system behind a large number of government open data initiatives;
  - such as Data.Gov, data.gov.au, and the European Data Portal
- Embedded JSON-LD fragments in HTML pages.
  - This is commonly used on many websites to describe their datasets.
  - Including many of those listed above.
  - But also Zenodo, Kaggle Datasets, all DataVerse sites and many others.

DataDepsGenerators.jl as the name suggests, generates static code which the user can add into their project's julia source code to make use of with DataDeps.jl. There are a number of reasons why static code generation is preferred over directly using the APIs. - On occasion the information reported by the APIs is wrong or incomplete. By generating code that the user may edit they may tweak the details as required. - The process of accessing the APIs requires a number of heavy dependencies, such as HTML and JSON parsers. If these APIs were to be access directly by a project, it would require adding this large dependency tree to the project. - It is important to know if a dataset has changed. As such retrieving the file hash and last modification date would be pointless if they are updated automatically. Finally: having the provenance information recorded in plain text, makes the dataset metadata readily accessible to anyone reading the source code; without having to run the project's application.

The automatic downloading of data is important to allow for robustly replicable scientific code. The inclusion of provenance information is required to give proper credit and to allow for good understanding of the dataset's real world context. DataDepsGenerators.jl makes this easy by automating most of the work.

B.1.1 Other similar packages

In the R software ecosystem there is the `suppdata` [@suppdata] package. `suppdata` is a package for easily downloading supplementary data files attached to journal articles. It is thus very similar in purpose: to make research data more accessible. It is a direct download tool, rather than DataDepsGenerators.jl's approach of generating metadata that is evaluated to preform the download. While there is some overlap, in that both support FigShare and Dryad, `suppdata` primarily supports journals rather than data repositories.

When it comes to accessing data repositories, there exists several R packages which only support a single provider of data. These vary in their support for different functionality. They often support things beyond the scope of DataDepsGenerators.jl, to search, or upload data to the supported repository. Examples include:

- `rdryad` for DataDryad
- `rfigshare` for FigShare
- `ckanr` for CKAN
- `rdatacite` for DataCite
- `rdataone` for DataOne

To the best of our knowledge at present their is not any unifying R package that supports anywhere near the range of data repostories supported by DataDepsGenerators.jl. Contemporaneously, with the creation of DataDepsGenerator.jl, the was proposed package to acquire data based on a DOI. While this has yet to eventuate into usable software, several of the discussions relating to it were insightful, and contributed to the functionality of DataDepsGenerators.jl.

To the best of our knowledge at present there does not exist a unifying R package that supports anywhere near the range of data repositories supported by DataDepsGenerators.jl. Contemporaneously, during the creation of DataDepsGenerator.jl, there was another R package (`doidata`) that was proposed in order to acquire data based on a DOI. While this has yet to eventuate into usable software, several of the discussions relating to it were insightful, and contributed to the functionality of DataDepsGenerators.jl.

B.1.2 Acknowledgements

This work was largely carried out as a Google Summer of Code project, as part of the NumFocus organisation. It also benefited from funding from Australian Research Council Grants DP150102405 and LP110100050.

We also wish to thank the support teams behind the APIs and repositories listed above. In the course of creating this tool we thoroughly exercised a number of APIs. In doing so we encountered a number of bugs and issues; almost all of which have now been fixed, by the attentive support and operation staff of the providers.

Appendix C

Embeddings.jl: easy access to pretrained word embeddings from Julia

This paper is currently under review for the Journal of Open Source Software.

C.1 Summary

Embeddings.jl is a tool to help users of the Julia programming language (Bezanson et al. 2014) make use of pretrained word embeddings for NLP. Word embeddings are a very important feature representation in natural language processing. The use of embeddings pretrained on very large corpora can be seen as a form of transfer learning. It allows knowledge of lexical semantics derived from the distributional hypothesis– that words occurring in similar contexts have similar meaning– to be injected into models which may have only limited amounts of supervised, task oriented training data.

Many creators of word embedding methods have generously made sets of pretrained word representations publicly available. Embeddings.jl exposes these as a standard matrix of numbers and a corresponding array of strings. This lets Julia programs use word embeddings easily, either on their own or alongside machine learning packages such as Flux (Innes 2018). In such deep learning packages, it is common to use word embeddings as an input layer of an LSTM or other neural network, where they may be kept invariant or used as initialization for fine-tuning on the supervised task. They can be summed to represent a bag of words, concatenated to form a matrix representation of a sentence or document, or used otherwise in a wide variety of natural language processing tasks.

Embeddings.jl makes use of DataDeps.jl (White et al. 2018), to allow for convenient automatic downloading of the data when and if required. It also uses the DataDeps.jl prompt to ensure the user of the embeddings has full knowledge of the original source of the data, and which papers to cite etc.

It currently provides access to

- multiple sets of word2vec embeddings (Mikolov et al. 2013) for English
- multiple sets of GLoVE (Pennington, Socher, and Manning 2014) embeddings for English
- multiple sets of FastText embeddings (Bojanowski et al. 2017; Grave et al. 2018) for several hundred languages

It is anticipated that as more pretrained embeddings are made available for more languages and using newer methods, the Embeddings.jl package will be updated to support them.

Appendix D

TensorFlow.jl: An Idiomatic Julia Front End for TensorFlow

This paper is currently under review for the Journal of Open Source Software.

D.1 Summary

TensorFlow.jl is a Julia (Bezanson et al. 2014) client library for the TensorFlow deep-learning framework (Abadi et al. 2015; Abadi et al. 2016). It allows users to define TensorFlow graphs using Julia syntax, which are interchangable with the graphs produced by Google's first-party Python TensorFlow client and can be used to perform training or inference on machine-learning models.

Graphs are primarily defined by overloading native Julia functions to operate on a TensorFlow.jl `Tensor` type, which represents a node in a TensorFlow computational graph. This overloading is powered by Julia's powerful multiple-dispatch system, which in turn allows allows the vast majority of Julia's existing array-processing functionality to work as well on the new `Tensor` type as they do on native Julia arrays. User code is often unaware and thereby reusable with respect to whtether its inputs are TensorFlow tensors or native Julia arrays by utilizing *duck-typing*.

TensorFlow.jl has an elegant, idiomatic Julia syntax. It allows all the usual infix operators such as `+`, `-`, `*` etc. It works seamlessly with Julia's broadcast syntax as well, such as the `.*` operator. This `*` can correspond to matrix multiplication while `.*` corresponds to element-wise multiplication, while Python clients needs distinct `@` (or `matmul`) and `*` (or `multiply`) functions. It also allows Julia-style indexing (e.g. `x[:, ii + end÷2]`), and concatenation (e.g. `[A B]`, `[x; y; 1]`). Its goal is to be idiomatic for Julia users, while still preserving all the power and maturity of the TensorFlow system. For example, it allows Julia code to operate on TPUs by virtue of using the same TensorFlow graph syntax as Python's TensorFlow client, even though there is no native Julia TPU compiler.

TensorFlow.jl to carefully balance between matching the Python TensorFlow API and Julia conventions. In turn, the Python TensorFlow client is itself designed to closely mirror numpy. Some examples are shown in the table below.

Julia	**Python TensorFlow**	**TensorFlow.jl**
1-based indexing	0-based indexing	1-based indexing
Column Major	Row Major	Row Major
explicit broadcasting	implicit broadcasting	implicit or explicit broadcasting
last index at `end` 2nd last in `end-1`	last index at `-1` second last in `-2`	last index at `end` 2nd last in `end-1`
Operations in Julia ecosystem namespaces. (`SVD` in `LinearAlgebra`, `erfc` in `SpecialFunctions`, `cos` in `Base`)	All operations TensorFlow's namespaces (`SVD` in `tf.linalg`, `erfc` in `tf.math`, `cos` in `tf.math`, and all reexported from `tf`)	All hand imported Operations in the Julia ecosystems namespaces. (`SVD` in `LinearAlgebra`, `erfc` in `SpecialFunctions`, `cos` in `Base`) Ops that have no other place are in `TensorFlow`. Automatically generated ops are in `Ops`
Container types are parametrized by number of dimensions and element type	N/A: does not have a parametric type system	Tensors are parametrized by element type, enabling easy specialization of algorithms for different types.

Defining TensorFlow graphs in the Python TensorFlow client can be viewed as metaprogramming, in the sense that a host language (Python) is being used to generate code in a different embedded language (the TensorFlow computational graph) (Innes et al. 2017). This often comes with some awkwardness, as the syntax and the semantics of the embedded language by definition do not match the host language or there would be no need for two languages to begin with. Using TensorFlow.jl is similarly a form of meta-programming for the same reason. However, the flexibility and meta-programming facilities offered by Julia's macro system makes Julia especially well-suited as a host language, as macros implemented in TensorFlow.jl can syntactically transform idiomatic Julia code into Julia code that constructs TensorFlow graphs. This permits users to reuse their knowledge of Julia, while users of the Python TensorFlow client essentially need to learn both Python and TensorFlow.

One example of our ability to leverage the increased expressiveness of Julia is using `@tf` macro blocks implemented in TensorFlow.jl to automatically name nodes in the TensorFlow computatioal graph. Nodes in a TensorFlow graph have names; these correspond to variable names in a traditional programming language. Thus every operation, variable and placeholder takes a `name` parameter. In most TensorFlow bindings, these must be specified manually resulting in a lot of code that includes duplicate information such as `x = tf.placeholder(tf.float32, name="x")` or they are defaulted to an uninformative value such as `Placeholder_1`. In TensorFlow.jl, prefixing a lexical block (such as a `function` or a `begin` block) with the `@tf` macro will cause the `name` parameter on all operations occurring on the right-hand side of an assignment to be filled in using the left-hand side. For example, the TensorFlow.jl equivalent of the above example is `@tf x = placeholder(Float32)`. Note how `x` is named only once instead of twice, as is redundantly required in the Python example. Since all nodes in the computational graph can automatically be assigned the same name as the corresponding Julia variable with no additional labor from TensorFlow.jl users, users get for free more intuitive debugging and graph visualisation.

to

Another example of the use of Julia's metaprogramming is in the automatic generation of Julia code for each operation defined by the official TensorFlow C implementation (for example, convolutions of two TensorFlow tensors). The C API can be queried to return definitions of all operations as protobuffer descriptions, which includes the expected TensorFlow type and arity of its inputs and outputs, as well as documentation. This described the operations at a sufficient level to generate the Julia code to bind to the functions in the C API and automatically generate a useful docstring for the function,. One challenge in this is that such generated code must correct the indices to be 1-based instead of 0-based to accord with Julia convention. Various heuristics are employed by TensorFlow.jl to guess which input arguments represent indicies and so should be converted.

TensorFlow.jl ships by default with bindings for most operations, but any operation can be dynamically imported at runtime using `@tfimport OperationName`, which will generate the binding and load it immediately. Additionally, for operations that correspond to native Julia operations (for example, `sin`), we overload the native Julia operation to call the proper binding.

We also use Julia's advanced parametric type system to enable elegant implementations of array operations not easily possible in other client libraries. TensorFlow.jl represents all nodes in the computational graph as parametric `Tensor` types which are parametrised by their element type, e.g. `Tensor{Int}`, `Tensor{Float64}` or `Tensor{Bool}`. This allows Julia's dispatch system to be used to simplify defining some bindings. For example, indexing a `Tensor` with an `Int`-like Tensor will ultimately create a node corresponding to a TensorFlow "gather" operation, and indexing with a `Bool`-like Tensor will correspond to a "boolean_mask" operation. It is also used to cast inputs in various functions to compatible shapes.

D.1.1 Challenges

A significant difficulty in implementing the TensorFlow.jl package for Julia is that in the upstream TensorFlow version 0 and 1 distributions, the C API is primarily designed for the execution of pretrained models and does not include many conveients for the definition of training of graphs.

The C API primarily exposes low-level array operations such as matrix multiplication or reductions. Graidnet descent ptimizers, RNNs functionality, and (until recently) shape-inference all required reimplementation on the Julia side. Most challengingly, the symbolic differentiation implemented in the `gradients` function is not available from the C API for all operations. To work around this, we currently use Julia's Python interop library to generate the gradient nodes using the Python client for those operations not supported by the C API. This requires serializing and deserializing TensorFlow graphs on both the Julia and Python side.

This has been improving over time, both due to Google moving more functionality from the Python TensorFlow client to the C API which can reused by Julia, and with more reimplementations of other aspects of the Python client from our own volunteer efforts. There nevertheless remains a large number of components from the upstream `contrib` submodule that remain unimplemented, including various efforts around probabilistic programming.

D.1.2 Other deep learning frameworks in Julia

Julia also has bespoke neural network packages such as Mocha (Zhang 2014), Knet (Yuret 2016) and Flux (Innes 2018), as well as bindings to other frameworks such as MxNet (Chen et al. 2015). While not having the full-capacity to directly leverage some of the benefits of the language and its ecosystem present in the pure julia frameworks such as Flux, TensorFlow.jl provides an interface to one of the most mature and widely deployed deep learning environments. It thus trivially supports technologies such as TPUs and visualization libraries like TensorBoard. It also gains the benefits from the any optimisations made in the graph execution engine of the underlying TensorFlow C

library, which includes extensive support for automatically distributing computations over multiple host machines which each have multiple GPUs.

D.1.3 Acknowledgements

- We gratefully acknowledge the 30 contributors to the TensorFlow.jl Github repository.
- We especially thank Katie Hyatt for contributing tests and documentation.
- We thank members of Julia Computing and the broader Julia Community for various discussions, especially Mike Innes and Keno Fischer.