dnadna: DEEP NEURAL ARCHITECTURES FOR DNA - A DEEP LEARNING FRAMEWORK FOR POPULATION GENETIC INFERENCE

Théophile Sanchez, Erik Madison Bray, Pierre Jobic, Jérémy Guez, Anne-Catherine Letournel, Guillaume Charpiat, Jean Cury, Flora Jay

DNADNA: DEEP NEURAL ARCHITECTURES FOR DNA - A DEEP LEARNING FRAMEWORK FOR POPULATION GENETIC INFERENCE

A PREPRINT

Théophile Sanchez [1*], **Erik Madison Bray** [1*],
Pierre Jobic [1,2], **Jérémy Guez** [1,3], **Anne-Catherine Letournel** [1], **Guillaume Charpiat** [1],
Jean Cury [1+], **Flora Jay** [1+]

[1] Université Paris-Saclay, CNRS UMR 9015, INRIA,
Laboratoire Interdisciplinaire des Sciences du Numérique, 91400, Orsay, France.
[2] ENS Paris-Saclay France.
[3] UMR7206 Eco-Anthropologie, Muséum National d'Histoire Naturelle, CNRS, Université de Paris, Paris, France.

[*] These authors contributed equally
[+] These authors contributed equally.
Corresponding authors:
flora.jay@lri.fr, jcury@normalesup.org

ABSTRACT

We present `dnadna`, a flexible python-based software for deep learning inference in population genetics. It is task-agnostic and aims at facilitating the development, reproducibility, dissemination, and reusability of neural networks designed for genetic polymorphism data.

`dnadna` defines multiple user-friendly workflows. First, users can implement new architectures and tasks, while benefiting from `dnadna` input/output and other utility functions, training procedure and test environment, which not only saves time but also decreases the probability of bugs. Second, implemented networks can be re-optimized based on user-specified training sets and/or tasks. Finally, users can apply pretrained networks in order to predict evolutionary history from alternative real or simulated genetic datasets, without the need of extensive knowledge in deep learning. Thanks to dnadna, newly implemented architectures and pretrained networks are easily shareable with the community for further benchmarking or applications.

`dnadna` comes with a peer-reviewed exchangeable neural network allowing demographic inference from SNP data, that can be used directly or retrained to solve other tasks. Toy networks are also available to ease the exploration of the software, and we expect that the range of available architectures will keep expanding thanks to contributions from the community.

Availability: `dnadna` repository is available at `https://gitlab.com/mlgenetics/dnadna` and its associated documentation at `https://mlgenetics.gitlab.io/dnadna/`.

Keywords population genetics · genomic data · deep neural networks · supervised learning · software

1 Introduction

Recent developments in deep learning methods for population genetics data and inference are flourishing. As reviewed by Sanchez *et al.* (2020), we distinguish two families of deep learning approaches for population genetics inference: those processing many summary statistics, with either fully connected or convolutional deep neural networks (e.g. Sheehan and Song, 2016; Mondal *et al.*, 2019; Xue *et al.*, 2021; Villanea and Schraiber, 2019; Montinaro *et al.*, 2021) and those, targeted by `dnadna`, based on 'raw' genetic data. The latter leverage deep neural networks to automatically construct informative features and bypass handcrafted summary statistics. Since 2008, a wide range of networks (fully-connected, convolutional, exchangeable convolutional, recurrent, adversarial, restricted Boltzmann machines, variational autoencoder) have been applied to multiple population genetic tasks, namely the inference of recombination or mutation rates (Chan *et al.*, 2018; Adrion *et al.*, 2020b; Burger *et al.*, 2021), various types of selection (Flagel *et al.*, 2019; Deelder *et al.*, 2021; Torada *et al.*, 2019; Gower *et al.*, 2021; Isildak *et al.*, 2021; Qin *et al.*, 2021), species delimitation (Derkarabetian *et al.*, 2019; Perez *et al.*, 2021), local or global introgression and ancestry (Flagel *et al.*, 2019; Montserrat *et al.*, 2020; Karim *et al.*, 2020; Wang *et al.*, 2021; Fonseca *et al.*, 2021; Meisner and Albrechtsen, 2020), past effective population sizes (Sanchez *et al.*, 2020; Wang *et al.*, 2021), geographic location (Battey *et al.*, 2020) and the visualization and/or generation of individual genomes (Yelmen *et al.*, 2021; Battey *et al.*, 2021; Montserrat *et al.*, 2019; Chen *et al.*, 2020), to name a few.

Previous studies provided their implementation at least for reproducibility and sometimes with a specific effort toward re-usability. Still, each of them focuses on a specific network for a specific task. These implementations can usually be adapted to newly simulated datasets and some hyperparameters can be modified. However, this requires first, an attentive reading and understanding of the code, and second, a direct modification of the code since many options are hard-coded. Not only is this subject to errors, but it also leads to a very rapid divergence of code between parallel projects and complex maintenance. We know the importance of having flexible and rigorous tools for the community, and this was again recently proven by the emergence of `stdpopsim`, a standardized library for population genetic simulations that provides thoroughly checked demographic scenarios, corresponding to previously inferred species histories, and allows contributions from many researchers (Adrion *et al.*, 2020a).

For these reasons we developed `dnadna`, Deep Neural Architectures for DNA, a comprehensive tool for population genetic inference. `dnadna` is a task-agnostic software that aims at facilitating applications, development, distribution and exchanges around neural networks in the population genetic community. `dnadna` notably enables researchers to (1) develop new networks or re-use existing architectures, (2) train them for a given task and (3) share them in such a way that users can easily apply these trained networks to their own dataset. Not only it is easy for users to reuse an existing network, but `dnadna` also helps in developing new architectures. For instance, many routines that may be otherwise overlooked, such as proper preprocessing or efficient loading of the data, are implemented and used seamlessly.

2 Software

`dnadna` is a python-based software providing utility functions and workflows for the development and application of neural networks in population genetics and is currently based on PyTorch. It implements the two main supervised machine learning tasks: regression and classification, and any tasks that would be a mix of those. In particular, it already implements several neural networks that were tested for inferring demographic and adaptive history from genetic data, and could be retrained on new simulations and/or to solve other tasks. Pre-trained networks can be used directly on real and simulated genetic polymorphism data for prediction. Implemented networks can also be optimized based on user-specified training sets or tasks. Finally, any user can implement new architectures and tasks, while benefiting from `dnadna` utility functions, training procedure, test environment, and the possibility to easily share this network with the community and facilitate benchmarking.

Figure 1 provides an overview of `dnadna` steps, all of which are independent. They are the classical steps required for any simulation-based supervised machine learning algorithm: (1) simulation, (2) preprocessing, (3) training, (4) predicting on test or real datasets. Note that the simulation step can be skipped if the user already possess a labelled dataset. Similarly, training can be skipped if the user reuses a pretrained network.

`dnadna` has a command line interface and an API. It is highly flexible thanks to a configuration file system based on the YAML format which provides the user a variety of options at each step of the process. We highlight here a few of them: (1a) for generating simulations: name of predefined scenario to be simulated and its related parameters, such as number of individuals, number of replicates, mutation rate, demographic parameters; (1b) for handling simulations: location on disk and filesystem layout; (2) for preprocessing: initial data transformations, filtering values such as minimal number of individuals or SNPs; (3a) for architecture design: network name and related arguments (number of filters, layers, ...);

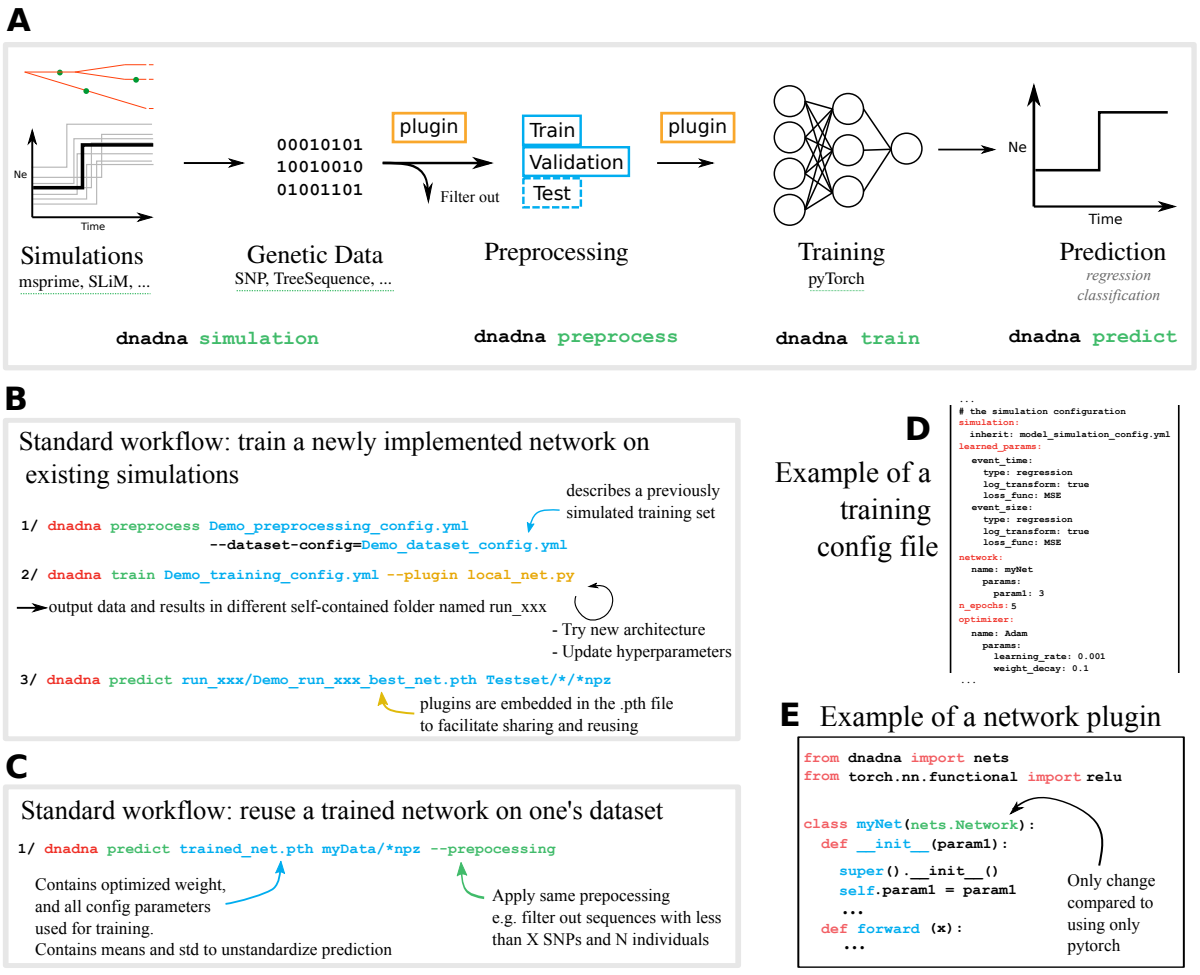

Figure 1: A: `dnadna` workflow and its corresponding commands. Each step could be done as a standalone. They are the classical steps required for any simulation-based supervised machine learning algorithm: (1) simulation of a large genetic dataset according to evolutionary scenarios and priors; (2) preprocessing, mainly to filter out examples that do not fit minimal requirements and split the rest into train/validation/test sets; (3) training neural networks; (4) predicting on test or real datasets using optimized neural networks. B and C: illustration of two standard use cases of `dnadna`. D: extract of a training configuration file in YAML format. E: view of a plugin python file, that will be passed to `dnadna train`, where users can implement novel networks based on PyTorch.

(3b) for training: loss functions, optimization and training hyperparameters (number of epochs, learning rate, batch size, optimizer name, ...); (3c) on-the-fly data transformations (subsampling, cropping, ...).

At each step several options are provided directly by `dnadna`. For instance, users can implement new simulators, new data transformations or new networks, other than what is already shipped with `dnadna`, without modifying the core of the code. By default, plugins are shared together with the optimized network (they are embedded in the same file), but they could also be merged into the code to benefit from maintenance, unit tests, and continuous integration to facilitate their adoption by the community.

For the exhaustive and up-to-date list of parameters, options and detailed instruction for each step of the process, the reader should refer to the online documentation `https://mlgenetics.gitlab.io/dnadna/`.

3 Tutorial examples

We provide multiple tutorials that will keep expanding. A Quickstart Tutorial walks the user through the complete `dnadna` process from configuring and generating simulated genetic data, to running data pre-processing on the simulated dataset, and training a network based off that dataset. A `msprime` Simulator Tutorial illustrates how to implement a simulation plugin, here based on `msprime` (Kelleher *et al.*, 2016), that can be integrated into `dnadna` (i.e. callable with `dnadna simulation` and outputting files in `dnadna` format). A notebook tutorial[1] presents an alternative where a few lines can be added into an existing `msprime` script to save the genetic data directly to the DNADNA format.

4 Implementation and distribution

`dnadna` is a Python (>=3.7) package with multiple dependencies, the main one being the open source machine learning library PyTorch. It has a command line interface and an API (`https://mlgenetics.gitlab.io/dnadna/api.html`). It is highly flexible thanks to a structured configuration file system based on YAML and JSON Schema.

`dnadna` is dual-licensed under the GNU Lesser General Public License (LGPLv3+) and the compatible CeCILL-C Free Software License Agreement (CECILL-C). Release 1.0 is available from PyPI at `https://pypi.org/project/dnadna/` and from Anaconda at `https://anaconda.org/mlgenetics/dnadna`. Docker images are available at `https://hub.docker.com/u/mlgenetics`.

For developers, we recommend cloning the GitLab repository (follow installation instructions). This also enables running the test suite via `pytest -v`. These tests are also run automatically when contributing to the GitLab project thanks to the continuous integration pipeline (tests are run both on CPU and GPU).

5 Conclusion

`dnadna` should allow researchers to focus on their research project, be it the analysis of population genetic data or building new methods, without the need to focus on proper development methodology (testing, continuous integration, documentation, etc.). Results will thus be more easily reproduced and shared. Having a common interface, instead of having many parallel projects, will limit the presence of bugs in the code. We emphasize that `dnadna` can be used both by users and developers of population genetic inference methods, and we encourage a culture of user-developers.

With this paper, we release `dnadna` version 1.0, which is a stable and usable version with many features, as described above. However, due to the complexity of developing deep learning methods for population genomics, and to the huge number of (hyper)parameters involved, some task or format may not be implemented yet. Also, many researchers in the deep learning community are not using PyTorch as a framework. We preferred it because it is deeply integrated to Python and thus, follows Python design which makes it easier to learn, read and write. It also benefits from a large community that is constantly developing new tools optimized with lower-level languages. But more importantly, we are working toward making `dnadna` framework-agnostic, where it is possible to use networks coded with TensorFlow or Keras with `dnadna`.

Overall `dnadna` is a substantial software with lots of ambition. We are eager for the deep learning and population genetics communities to embrace it and to provide feedback and contributions to it.

[1] `https://gitlab.com/mlgenetics/dnadna/-/blob/master/examples/`

Acknowledgements

TAU GPU platform (Titanic) and Kepler for computing resources. ASARD team (LISN), SADL LRI. DIM One Health 2017 (number RPH17094JJP), Human Frontier Science Project (number RGY0075/2019), Paris-Saclay Center for Data Science 2.0 (IRS), ANR-80Prime TransIA, ANR-20-CE45-0010-01 RoDAPoG for funding.

References

Adrion, J. R. *et al.* (2020a). A community-maintained standard library of population genetic models. *eLife*, **9**, e54967. Publisher: eLife Sciences Publications, Ltd.

Adrion, J. R. *et al.* (2020b). Predicting the Landscape of Recombination Using Deep Learning. *Molecular Biology and Evolution*, **37**(6), 1790–1808.

Battey, C. *et al.* (2020). Predicting geographic location from genetic variation with deep neural networks. *eLife*, **9**, e54507. Publisher: eLife Sciences Publications, Ltd.

Battey, C. J. *et al.* (2021). Visualizing population structure with variational autoencoders. *G3 Genes|Genomes|Genetics*, **11**(1), 1–11.

Burger, K. E. *et al.* (2021). Neural Networks for self-adjusting Mutation Rate Estimation when the Recombination Rate is unknown. preprint, Evolutionary Biology.

Chan, J. *et al.* (2018). A Likelihood-Free Inference Framework for Population Genetic Data using Exchangeable Neural Networks. *Advances in Neural Information Processing Systems*, **31**, 8594–8605.

Chen, J. *et al.* (2020). Population-scale Genomic Data Augmentation Based on Conditional Generative Adversarial Networks. In *Proceedings of the 11th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, pages 1–6, Virtual Event USA. ACM.

Deelder, W. *et al.* (2021). Using deep learning to identify recent positive selection in malaria parasite sequence data. *Malaria Journal*, **20**(1), 270.

Derkarabetian, S. *et al.* (2019). A demonstration of unsupervised machine learning in species delimitation. *Molecular Phylogenetics and Evolution*, **139**, 106562.

Flagel, L. *et al.* (2019). The Unreasonable Effectiveness of Convolutional Neural Networks in Population Genetic Inference. *Molecular Biology and Evolution*, **36**(2), 220–238.

Fonseca, E. M. *et al.* (2021). Phylogeographic model selection using convolutional neural networks. *Molecular Ecology Resources*, **21**(8), 2661–2675. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1755-0998.13427.

Gower, G. *et al.* (2021). Detecting adaptive introgression in human evolution using convolutional neural networks. *eLife*, **10**, e64669. Publisher: eLife Sciences Publications, Ltd.

Isildak, U. *et al.* (2021). Distinguishing between recent balancing selection and incomplete sweep using deep neural networks. *Molecular Ecology Resources*, **21**(8), 2706–2718. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1755-0998.13379.

Karim, M. R. *et al.* (2020). Convolutional Embedded Networks for Population Scale Clustering and Bio-ancestry Inferencing. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pages 1–1. Conference Name: IEEE/ACM Transactions on Computational Biology and Bioinformatics.

Kelleher, J. *et al.* (2016). Efficient Coalescent Simulation and Genealogical Analysis for Large Sample Sizes. *PLOS Computational Biology*, **12**(5), e1004842.

Meisner, J. and Albrechtsen, A. (2020). Haplotype and Population Structure Inference using Neural Networks in Whole-Genome Sequencing Data. preprint, Bioinformatics.

Mondal, M. *et al.* (2019). Approximate Bayesian computation with deep learning supports a third archaic introgression in Asia and Oceania. *Nature Communications*, **10**(1).

Montinaro, F. *et al.* (2021). Revisiting the out of Africa event with a deep-learning approach. *The American Journal of Human Genetics*, **108**(11), 2037–2051.

Montserrat, D. M. *et al.* (2019). Class-Conditional VAE-GAN for Local-Ancestry Simulation. *arXiv:1911.13220 [cs, q-bio, stat]*. arXiv: 1911.13220.

Montserrat, D. M. *et al.* (2020). LAI-Net: Local-Ancestry Inference with Neural Networks. *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1314–1318. arXiv: 2004.10377.

Perez, M. F. *et al.* (2021). Coalescent-based species delimitation meets deep learning: Insights from a highly fragmented cactus system. *Molecular Ecology Resources*, **n/a**(n/a).

Qin, X. *et al.* (2021). Deciphering signatures of natural selection via deep learning. preprint, Genomics.

Sanchez, T. *et al.* (2020). Deep learning for population size history inference: Design, comparison and combination with approximate Bayesian computation. *Molecular Ecology Resources*. Publisher: Wiley/Blackwell.

Sheehan, S. and Song, Y. S. (2016). Deep Learning for Population Genetic Inference. *PLOS Computational Biology*, **12**(3), e1004845.

Torada, L. *et al.* (2019). ImaGene: a convolutional neural network to quantify natural selection from genomic data. *BMC Bioinformatics*, **20**(S9), 337.

Villanea, F. A. and Schraiber, J. G. (2019). Multiple episodes of interbreeding between Neanderthal and modern humans. *Nature Ecology & Evolution*, **3**(1), 39–44.

Wang, Z. *et al.* (2021). Automatic inference of demographic parameters using generative adversarial networks. *Molecular Ecology Resources*, **n/a**(n/a). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1755-0998.13386.

Xue, A. T. *et al.* (2021). Discovery of Ongoing Selective Sweeps within Anopheles Mosquito Populations Using Deep Learning. *Molecular Biology and Evolution*, **38**(3), 1168–1183.

Yelmen, B. *et al.* (2021). Creating artificial human genomes using generative neural networks. *PLOS Genetics*, **17**(2), e1009303. Publisher: Public Library of Science.