The Yin-Yang dataset

Laura Kriener
laura.kriener@unibe.ch
Department of Physiology,
University of Bern
Switzerland

Julian Göltz
julian.goeltz@kip.uni-heidelberg.de
Kirchhoff-Institute for Physics,
Heidelberg University
Germany
Department of Physiology,
University of Bern
Switzerland

Mihai A. Petrovici
mihai.petrovici@unibe.ch
Department of Physiology,
University of Bern
Switzerland
Kirchhoff-Institute for Physics,
Heidelberg University
Germany

ABSTRACT

The Yin-Yang dataset was developed for research on biologically plausible error backpropagation and deep learning in spiking neural networks. It serves as an alternative to classic deep learning datasets, especially in early-stage prototyping scenarios for both network models and hardware platforms, for which it provides several advantages. First, it is smaller and therefore faster to learn, thereby being better suited for small-scale exploratory studies in both software simulations and hardware prototypes. Second, it exhibits a very clear gap between the accuracies achievable using shallow as compared to deep neural networks. Third, it is easily transferable between spatial and temporal input domains, making it interesting for different types of classification scenarios.

CCS CONCEPTS

• **Networks** → **Network performance evaluation**; • **Computing methodologies** → *Bio-inspired approaches*; • **Hardware** → **Functional verification**.

ACM Reference Format:
Laura Kriener, Julian Göltz, and Mihai A. Petrovici. 2022. The Yin-Yang dataset. In *Neuro-Inspired Computational Elements Conference (NICE 2022), March 28-April 1, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3517343.3517380

1 INTRODUCTION

We introduce the Yin-Yang dataset for learning in hierarchical networks [11]. It is tailored to the requirements of research on biologically plausible error backpropagation algorithms, learning in spiking neural networks and hierarchical networks on neuromorphic hardware. These fields typically require small but at the same time not trivially solvable datasets to prototype and test network architectures and learning algorithms. Setups commonly used for this purpose involve either elementary logic tasks such as XOR or small-scale datasets such as MNIST or fashion-MNIST [12, 22] and reduced versions thereof. However, these setups often do not adequately fulfill their purpose. Binary XOR only has a tiny number of input patterns and therefore a very limited, discrete set of reachable accuracies, making the evaluation and comparison of learning algorithms difficult. In turn, MNIST-type datasets have other drawbacks. For one, they require comparatively large networks, which might not be feasible during prototyping. But even more importantly, and despite this ostensible difficulty, they can nevertheless be classified with high accuracy even by shallow networks or networks without learning in the lower layers. This is problematic because training a deep network with an imperfect learning algorithm can result in performance indistinguishable from that of a shallow network or a network with plasticity only in the last layer. Conversely, a test on the MNIST dataset can fail to reveal the inability of the training algorithm to propagate error signals through the network, as the achieved high accuracies obscure the underlying problem.

The Yin-Yang dataset can provide an alternative for these testing and prototyping scenarios as it is solvable by smaller networks, contains fewer samples and most importantly exhibits a large gap between the accuracies reached by shallow or partly fixed networks on the one hand and correctly trained deep networks on the other. Note that here, we use "deep" in opposition to "shallow", i.e., any network that has latent variables through which errors need to propagate. We consider a shallow network to be the equivalent of a single-layer perceptron, with only an input layer connected directly to a label layer.

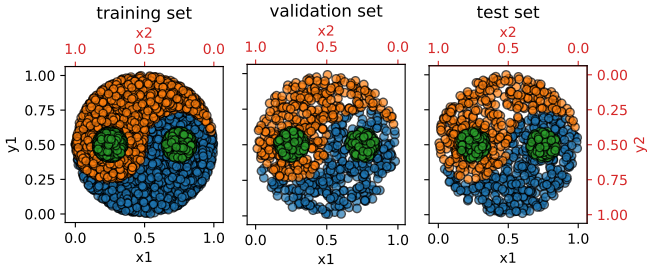

Figure 1: Training, validation and test dataset. Each dot in the yin-yang symbol represents one sample of the dataset. The color of the dot denotes its class ("Yin", "Yang" or "Dot"). This figure was generated using the default settings for random seeds and dataset sizes (5000 samples for the training set and 1000 samples each for the validation and test set).

NICE 2022, March 28-April 1, 2022, Virtual Event, USA

ACM ISBN 978-1-4503-9559-5/22/03.
https://doi.org/10.1145/3517343.3517380

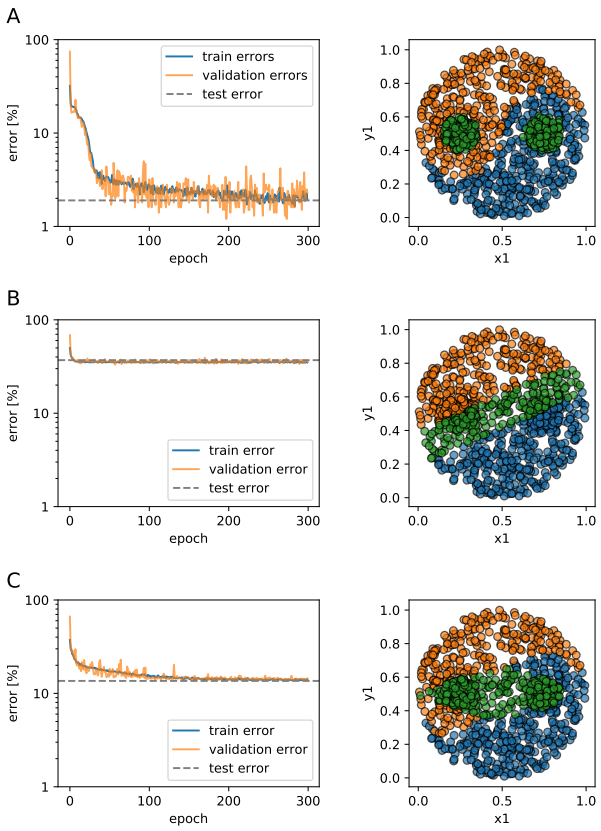

Figure 2: Comparison of exemplary training results for different network setups. Network parameters are given in Table 2. Left column: Evolution of the validation and training error during training. Right column: training result illustrated on the test set. (A) Network with one hidden layer and fully functional synaptic plasticity via classical error backpropagation. (B) Shallow network. (C) Network with one hidden layer and frozen lower weights.

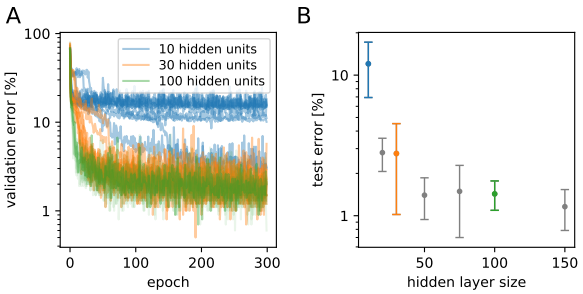

Figure 3: Impact of hidden layer size on network performance. (A) Validation errors during training for three different network architectures with different hidden layer sizes. For each architecture, ten training runs with different random weight initialization are overlaid. (B) Mean and standard deviation of the final test error depending on hidden layer size of the network. The colored data points correspond to the runs shown in A.

2 DATASET

Each sample in the dataset represents a point in a two-dimensional representation of the yin-yang symbol. Depending on their location in the symbol the samples are classified into the "Yin", "Yang" or "Dot" class (Fig. 1). Even though the areas in the yin-yang symbol covered by the different classes have different sizes, the dataset is designed to be balanced, which means that all classes are represented by approximately the same amount of samples. Note that therefore the density of samples is higher in the "Dot"-class regions, as the combined area of these regions is smaller than that of the others.

The samples are randomly generated using rejection sampling. The exact version of a generated set of samples is therefore determined by the random seed and dataset size. This makes it possible to produce multiple dataset versions by providing different random seeds and dataset sizes. In the default configuration the training set has 5000 samples while the validation and test sets have 1000 samples respectively, each generated with a different random seed.

As can be seen in Fig. 1, values of all samples in the dataset are strictly positive. This is the case to accommodate network models which require positive input values only (common in the field of biologically-plausible networks, as firing rates as well as spike times are typically denoted by positive numbers). Because of that the yin-yang symbol is not centered around zero. This however complicates training in neuron models without intrinsic (learnable) bias. To facilitate training for these models, each sample in the dataset consists not only of the coordinates (x, y) determining the position in the yin-yang symbol but additionally also the values $(1 - x, 1 - y)$. This effectively symmetrizes the input and removes the need for a bias even though the yin-yang symbol is not centered around the origin of the coordinate system.

3 TRAINING RESULTS

As a baseline for further applications of this dataset we also provide some training results achieved with classical artificial neural networks. In particular, we compare network performance in three scenarios:

(1) a network with one hidden layer and fully functional error backpropagation;
(2) a shallow network with only an input and an output layer;
(3) a network with one hidden layer, but with frozen weights between the input and the hidden layer to emulate training with a faulty error backpropagation algorithm.

For all scenarios, we use very small network sizes to emulate a model or hardware prototyping environment. Incidentally, this is also helpful in highlighting another problem that is frequently overlooked when increasing the network size: because a large enough hidden layer can mask faulty error backpropagation, larger-scale networks are often inadequate for a quantitative verification of credit assignment (precise error propagation) within the studied network model. This is discussed below in more detail.

Table 1: Mean and standard deviation of the test accuracy for 20 training runs with different random initializations for different network configurations. Training parameters can be found in Table 2.

network	hidden layer with 20 neurons	hidden layer with 30 neurons
deep network	(97.0 ± 1.6) %	(97.6 ± 1.5) %
deep network (frozen lower weights)	(78.3 ± 7.8) %	(85.5 ± 5.8) %
shallow network	(63.8 ± 1.0) %	

The comparison between the three scenarios (Table 1 and Fig. 2) illustrates a manifest advantage of the Yin-Yang dataset compared to other commonly used datasets of comparable size: both the shallow network and the one with the frozen lower weights are clearly unable to learn the required features to successfully classify the dataset. This leads to a gap of more than 30 % between the accuracies achieved by a shallow and a deep network.

The failure of the partially frozen network highlights another important issue for various proposals of bio-plausible solutions to the credit assignment problem. In large enough networks, the large hidden layers project the input into a very high-dimensional space, which makes classification tasks more easily solvable by the linear classifier embodied by the top layer. This is commonly referred to as the "kernel trick" (see e.g. [19]). This can easily mask the inability of a network to correctly propagate errors and perform true gradient descent learning. While this issue would become observable when dealing with more complicated classification problems, it would require using large, deep networks that are not only difficult to debug but, more importantly, would lie beyond the capabilities of typical prototype devices or software simulations.

The Yin-Yang dataset addresses both problems simultaneously, by clearly highlighting faulty error backpropagation already within resource-efficient implementations with hidden layer sizes of around 20 to 30 neurons (see Table 1). Under these circumstances, the difference between the accuracy reached by a properly trained network and the network where only the top weights are trained lies around 20 % and 12 % respectively. This is a much higher gap than in a comparable example with the MNIST dataset, where networks need several hundred hidden neurons to show significant performance improvements beyond linear classifiers [12]. However, such sizes automatically introduce the kernel trick: a network with 500 hidden units reaches on average 98.3 % on MNIST, while the same network with only training in the top layer reaches 94.8 %. Unmasking these issues can become crucial in research on biologically plausible forms of credit assignment and (local) synaptic plasticity, where exact error backpropagation is notoriously difficult to realize, both for rate-based models and, even more pronouncedly, for spiking networks.

Another advantage of the Yin-Yang dataset over many other commonly used datasets is the dimensionality of its samples and the network sizes required to learn the task. Each sample consists of only four input values (compared to, e.g., the 784 input channels required by MNIST), which significantly reduces the required fan-in for hidden neurons. This can be especially beneficial on neuromorphic platforms, where the number of synaptic connections to a neuron is very often limited by the chip architecture, even more so for early-stage prototypes (e.g. [2, Section 3.3], [1, 5, 13, 14, 17]).

Also, this dataset can be learned with a single hidden layer of reasonably small size (Fig. 3). For consistently high final accuracies, a hidden layer of 20 to 30 neurons is required, but for a small proof-of-concept demonstration of a learning algorithm or hardware prototype, even 10 hidden units are enough to achieve results (around 88 % accuracy) that would be impossible with shallow networks, or with algorithms that cannot profit from a network's representational hierarchy. The full set of training parameters can be found in Table 2.

In addition to the results shown here, the dataset has already been used to showcase algorithms for error backpropagation in spiking neural networks in [7] and [21].

Table 2: Training parameters used to produce the results in Fig. 2. Fig. 3 uses the same parameters except for the size of the hidden layer.

parameter name	value
activation function	ReLU
size input	4
size hidden layer (for deep net)	30
size output layer	3
training epochs	300
batch size	20
optimizer	Adam, [10]
Adam parameter β	$(0.9, 0.999)$
Adam parameter ϵ	10^{-8}
learning rate	0.01

4 INPUT ENCODING

The Yin-Yang dataset can be adapted to suit the needs of very different network models. Depending on the used network architecture, neuron model and mode of communication between the neurons, different types of information encoding become necessary. In the following, we discuss several encoding methods that are well-suited for a variety of different network and neuron types.

4.1 Spatio-temporal input encoding

Using this dataset for spiking neural networks requires an explicit spatio-temporal input encoding. In [7] and [21], the four input features of the dataset were directly interpreted as the spike times of 4 input neurons (Fig. 4 A). This was done by choosing parameters t_{early} and t_{late} as the earliest and latest possible time the input neurons are allowed to spike. Then the dataset values $\vec{x} = (x, y, 1 -$

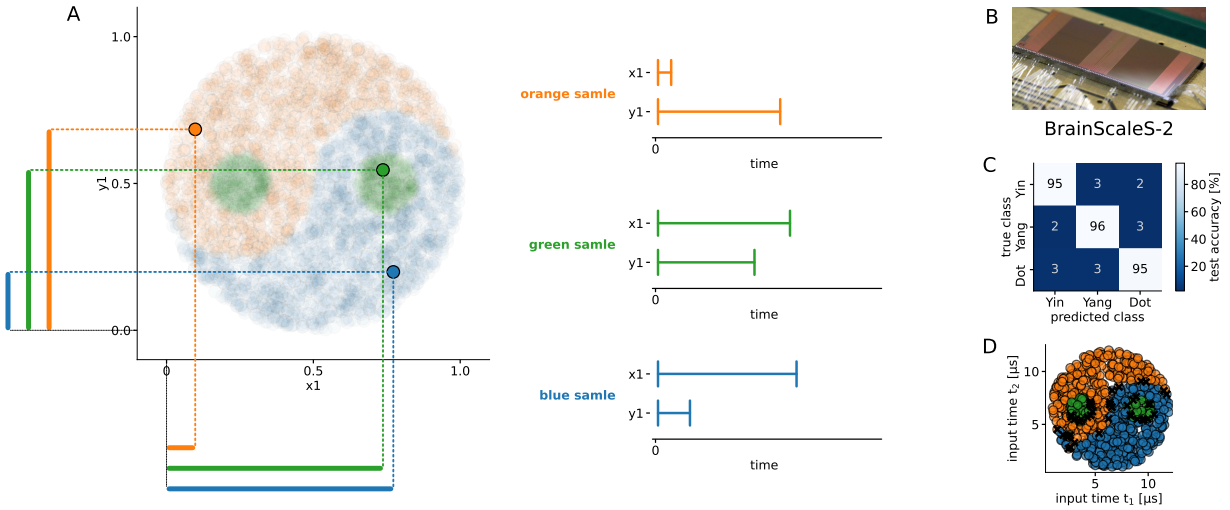

Figure 4: Spatio-temporal input encoding scheme and classification results on the neuromorphic chip BrainScaleS-2. Panels (B-D) adapted from [6]. (A) Encoding of the x, y-coordinates of the Yin-Yang pattern as input spike times t_1 and t_2 illustrated on one sample each for the three classes. (B) Image of the BrainScaleS-2 ASIC. (C) Confusion matrix after training the BrainScaleS-2 chip to classify the Yin-Yang dataset. (D) Classification result of the chip on the test set. For each input sample the color indicates the class determined by the trained network. Wrong classifications are marked with a black X. The wrongly classified samples all lie very close to the border between two classes.

$x, 1-y$) were translated into the four spike times $\vec{t} = (t_1, t_2, t_3, t_4)$ as follows:

$$\vec{t} = t_{\text{early}} + \vec{x} \cdot \left(t_{\text{late}} - t_{\text{early}}\right) \tag{1}$$

The choice of $t_{\text{early/late}}$ is dependent on the network architecture and employed learning algorithms. For [7] it has proven beneficial to choose t_{early} slightly after the start of the experiment and t_{late} as the sum of the two neuron time constants $t_{\text{late}} \approx \tau_{\text{m}} + \tau_{\text{syn}}$. The classification results achieved with the BrainScaleS-2 chip are shown in Fig. 4.

Alternatively, a different spike-based spatio-temporal encoding can be achieved implicitly by manipulating input currents, as proposed for example in [3]. Here, each input variable is interpreted as the strength of a constant input current into a leaky-integrate and fire neuron. The timing of the output spike of the input neurons depends on the strength of the input current I with

$$t_{\text{spike}} = \tau_m \log \frac{I}{I - \theta_{\text{I}}} \tag{2}$$

where τ_m denotes the membrane time constant and θ_{I} the minimal current necessary to evoke an output spike.

4.2 Rate-based input encoding

Many models for biologically plausible error backpropagation are built around rate-based neuron models (e.g. [9, 15, 16], for a review see also [20]). These approaches use continuous rates as an idealized version of rate coding in spiking neurons. Others build on the same approximations but explicitly use spike-based communication in their neural network implementations (e.g. [4, 8, 18]). For such rate-based models, a suitable encoding scheme can be easily realized by designating 4 input neurons and setting their output rates proportional to the values of the respective input feature.

In case of spiking neurons, these four input neurons can produce Poisson spike trains with the same rates as their rate-based counterparts, as, for example, in [18]. Alternatively, regular spike trains could also be used to represent firing rates; while more precise than the intrinsicly stochastic Poisson solution, this scheme has its own potential drawback of making the neuronal input-output function dependent on not just the rate, but also the phase of a neuron's afferents. Under certain circumstances, encoding an input as a single neuron may not be viable, for example when synaptic bandwidth or neuron firing rate are limited. In this case, one input can be represented by a population of neurons with a mean firing rate equal to the value of the input.

Code and data availability

Code for the Yin-Yang data set is available at https://github.com/lkriener/yin_yang_data_set. The example notebook in the repository includes the plotting of the data samples (Fig. 1) and the training of deep and shallow networks (Fig. 2). Additional data available on request from the authors.

REFERENCES

[1] Sebastian Billaudelle, Yannik Stradmann, Korbinian Schreiber, Benjamin Cramer, Andreas Baumbach, Dominik Dold, Julian Göltz, Akos F Kungl, Timo C Wunderlich, Andreas Hartel, et al. 2020. Versatile emulation of spiking neural networks on an accelerated neuromorphic substrate. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.

[2] Jonathan Binas, Daniel Neil, Giacomo Indiveri, Shih-Chii Liu, and Michael Pfeiffer. 2016. Precise neural network computation with imprecise analog devices. *arXiv preprint arXiv:1606.07786* (2016).

[3] Benjamin Cramer, Sebastian Billaudelle, Simeon Kanya, Aron Leibfried, Andreas Grübl, Vitali Karasenko, Christian Pehle, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, et al. 2020. Surrogate gradients for analog neuromorphic computing. *arXiv preprint arXiv:2006.07239* (2020).
[4] Steve K Esser, Rathinakumar Appuswamy, Paul Merolla, John V Arthur, and Dharmendra S Modha. 2015. Backpropagation for energy-efficient neuromorphic computing. *Advances in Neural Information Processing Systems* (2015), 1117–1125.
[5] Charlotte Frenkel, Martin Lefebvre, Jean-Didier Legat, and David Bol. 2018. A 0.086-mm ^{2} 12.7-pJ/SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS. *IEEE transactions on biomedical circuits and systems* 13, 1 (2018), 145–158.
[6] Julian Göltz, Laura Kriener, Andreas Baumbach, Sebastian Billaudelle, Oliver Breitwieser, Benjamin Cramer, Dominik Dold, Akos Ferenc Kungl, Walter Senn, Johannes Schemmel, Karlheinz Meier, and Mihai A Petrovici. 2019. Fast and deep: energy-efficient neuromorphic learning with first-spike times. *arXiv:1912.11443* (2019).
[7] Julian Göltz, Laura Kriener, Andreas Baumbach, Sebastian Billaudelle, Oliver Breitwieser, Benjamin Cramer, Dominik Dold, Akos Ferenc Kungl, Walter Senn, Johannes Schemmel, Karlheinz Meier, and Mihai A Petrovici. 2021. Fast and energy-efficient neuromorphic deep learning with first-spike times. *Nature Machine Intelligence* 3, 9 (2021), 823–835.
[8] Jordan Guerguiev, Timothy P Lillicrap, and Blake A Richards. 2017. Towards deep learning with segregated dendrites. *Elife* 6 (2017), e22901.
[9] Paul Haider, Benjamin Ellenberger, Laura Kriener, Jakob Jordan, Walter Senn, and Mihai A Petrovici. 2021. Latent Equilibrium: Arbitrarily fast computation with arbitrarily slow neurons. *Advances in Neural Information Processing Systems* 34 (2021).
[10] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980* (2014).
[11] Laura Kriener, Julian Göltz, and Mihai A Petrovici. 2021. Yin-Yang dataset repository. https://github.com/lkriener/yin_yang_data_set. Accessed: 2021-01-20.
[12] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[13] Saber Moradi and Giacomo Indiveri. 2013. An event-based neural network architecture with an asynchronous programmable synaptic memory. *IEEE transactions on biomedical circuits and systems* 8, 1 (2013), 98–107.
[14] Manu V Nair and Giacomo Indiveri. 2019. An ultra-low power sigma-delta neuron circuit. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.
[15] João Sacramento, Rui Ponte Costa, Yoshua Bengio, and Walter Senn. 2018. Dendritic cortical microcircuits approximate the backpropagation algorithm. *Advances in Neural Information Processing Systems* 31 (2018).
[16] Benjamin Scellier and Yoshua Bengio. 2017. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience* 11 (2017), 24.
[17] Johannes Schemmel, Laura Kriener, Paul Müller, and Karlheinz Meier. 2017. An accelerated analog neuromorphic hardware system emulating NMDA-and calcium-based non-linear dendrites. In *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2217–2226.
[18] Sebastian Schmitt, Johann Klähn, Guillaume Bellec, Andreas Grübl, Maurice Güttler, Andreas Hartel, Stephan Hartmann, Dan Husmann, Kai Husmann, Sebastian Jeltsch, et al. 2017. Neuromorphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system. *2017 International Joint Conference on Neural Networks (IJCNN)* (2017), 2227–2234.
[19] Bernhard Scholkopf. 2001. The kernel trick for distances. *Advances in neural information processing systems* (2001), 301–307.
[20] James CR Whittington and Rafal Bogacz. 2019. Theories of error back-propagation in the brain. *Trends in cognitive sciences* 23, 3 (2019), 235–250.
[21] Timo C Wunderlich and Christian Pehle. 2021. Event-based backpropagation can compute exact gradients for spiking neural networks. *Scientific Reports* 11, 1 (2021), 1–17.
[22] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv:1708.07747* (2017).

ACKNOWLEDGMENTS

We wish to thank Sebastian Billaudelle and Benjamin Cramer for valuable discussions, as well as Mike Davies and Intel for their ongoing support. We gratefully acknowledge funding from the European Union under grant agreements 604102, 720270, 785907, 945539 (HBP) and the Manfred Stärk Foundation.

During the development of the dataset some calculations were performed on UBELIX, the HPC cluster at the University of Bern, others were performed on the bwForCluster NEMO, supported by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant no INST 39/963-1 FUGG. Additionally, our work has greatly benefitted from access to the Fenix Infrastructure resources, which are partially funded from the European Union's Horizon 2020 research and innovation programme through the ICEI project under the grant agreement No. 800858.