Bioinformatics, 38(9), 2022, 2381–2388
https://doi.org/10.1093/bioinformatics/btac101
Advance Access Publication Date: 22 February 2022
Original Paper

Genome analysis

seqgra: principled selection of neural network architectures for genomics prediction tasks

Konstantin Krismer [1,2], Jennifer Hammelman [1,3] and David K. Gifford [1,2,3,4,]*

[1]Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, USA, [2]Department of Biological Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, USA, [3]Computational and Systems Biology, Massachusetts Institute of Technology, Cambridge, MA 02139, USA and [4]Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

*To whom correspondence should be addressed.

Associate Editor: Valentina Boeva

Received on June 15, 2021; revised on January 26, 2022; editorial decision on February 12, 2022; accepted on February 18, 2022

Abstract

Motivation: Sequence models based on deep neural networks have achieved state-of-the-art performance on regulatory genomics prediction tasks, such as chromatin accessibility and transcription factor binding. But despite their high accuracy, their contributions to a mechanistic understanding of the biology of regulatory elements is often hindered by the complexity of the predictive model and thus poor interpretability of its decision boundaries. To address this, we introduce seqgra, a deep learning pipeline that incorporates the rule-based simulation of biological sequence data and the training and evaluation of models, whose decision boundaries mirror the rules from the simulation process.

Results: We show that seqgra can be used to (i) generate data under the assumption of a hypothesized model of genome regulation, (ii) identify neural network architectures capable of recovering the rules of said model and (iii) analyze a model's predictive performance as a function of training set size and the complexity of the rules behind the simulated data.

Availability and implementation: The source code of the seqgra package is hosted on GitHub (https://github.com/gifford-lab/seqgra). seqgra is a pip-installable Python package. Extensive documentation can be found at https://kkrismer.github.io/seqgra.

Contact: gifford@mit.edu

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Over the last 5–10 years, neural networks were successfully applied to make large gains on a wide range of tasks in such diverse fields as computer vision, computer audition, natural language processing and robotics. While the structure and the semantics of the data used to train and evaluate neural networks can be vastly different, the core learning algorithms are almost always the same and the neural network architectures are often composed of similar building blocks. This is also true for the field of genomics, and computational biology as a whole, where deep neural networks are trained on data that are obtained experimentally using functional genomics assays such as DNase-seq (Boyle *et al.*, 2008), ATAC-seq (Buenrostro *et al.*, 2013) and ChIP-seq. Motivated by their success, architectural building blocks commonly seen in these networks, such as convolutional layers, recurrent layers, batch normalization, dropout and skip connections (Kelley *et al.*, 2016; Nair *et al.*, 2019; Quang and Xie, 2016; Zhou and Troyanskaya, 2015), have been imported from computer vision and other fields. This cross-fertilization between fields and the general applicability of the building blocks of deep learning has more recently been seen in the adoption of transformer-based architectures for image classification tasks in computer vision and protein prediction tasks in biology. However, most datasets used to train supervised deep learning models in biology are different from datasets in computer vision and natural language processing in two ways. (i) Biological problems contain noisy input and noisy labels in that not only is there substantial intraclass variability and noise in the input, e.g. images labeled as *cat* contain cats that vary in terms of breed, color, position, pose, etc., but also a significant fraction of examples are mislabeled, i.e. images labeled as *cat* are empty or contain dogs. This is rare in computer vision datasets, but common in datasets derived from functional genomics assays. (ii) Feature attribution or other model explanation methods are not human-interpretable. We understand images of cats in the sense that we know which parts of the image contain information that is relevant for the classification (because they belong to the cat) and which parts are irrelevant (because they belong to the background). This intuitive understanding is necessary when attribution methods such as saliency maps are applied to assess a model's ability to base predictions on relevant parts of the input. In biology, examples often include DNA sequence windows of various widths, most commonly 1000 base pairs (bp), which, unlike images of cats, are not *human-readable*. This biology-specific issue of inherently opaque examples exacerbates the general interpretability issue of deep neural networks, whereas the lack of high-quality datasets contributes to the reproducibility crisis and makes it more difficult to compare architectures, as they are often only evaluated on a custom dataset.

The method introduced here, *seqgra*, attempts to improve the process by which neural network architectures are chosen for specific genomics prediction tasks and provides a framework to evaluate model interpretation methods. Its fully reproducible pipeline provides a means to (i) simulate data based on a predefined set of probabilistic rules, (ii) create and train models based on a precise description of their architecture, loss, optimizer and training process and (iii) evaluate the trained models using conventional test set metrics as well as an array of feature attribution methods. These feature attribution methods in combination with simulated data and thus perfect ground truth enable an analysis of the model's decision boundaries and how well they capture the underlying rules of the data generation process from step 1. Utilizing this framework, models are not only evaluated based on their predictive performance, but also on the ability to recover the vocabulary (e.g. specific transcription factor binding site motifs) and grammar (e.g. spacing constraints between interacting transcription factors) of the dataset, while assigning little weight to confounding factors and idiosyncratic noise.

Efforts in this area include Kipoi (Avsec *et al.*, 2019), a repository for trained genomics models, and Selene (Chen *et al.*, 2019), a framework for biological sequence-based deep learning models that supports training of PyTorch models, model evaluation with conventional test set metrics (ROC and precision–recall curves), and variant effect prediction and *in silico* mutagenesis of trained models. To our knowledge none of the existing methods offer functionality for simulating data using a general framework of probabilistic rules, nor do they incorporate feature attribution methods.

Furthermore, this simulation-based framework can also serve as a means to investigate the strengths and weaknesses of various feature attribution methods across different neural network architectures that are trained on datasets with varying degrees of complexity. With simulated and thus perfect data, the idiosyncrasies of attribution methods can more easily be exposed.

2 Materials and methods

2.1 Alphabet distribution for grammars

For all grammars discussed in this paper, we used the natural nucleotide distribution of the human genome, 29.565% adenine (A), 20.435% cytosine (C), 20.435% guanine (G) and 29.565% thymine (T) (Piovesan *et al.*, 2019).

2.2 Motif database

We used HOMER motifs for all grammar sequence elements that were based on transcription factor binding site motifs. These motifs were obtained by analyzing data from publicly available ChIP-seq experiments (Heinz *et al.*, 2010).

2.3 Feature importance evaluators

While conventional test set metrics, such as ROC curves and precision–recall curves, assess model performance based on a set of examples (e.g. the test set), feature importance evaluators (FIEs) quantify the contribution of each input feature to the model's prediction. In the context of seqgra, FIEs are used to assess what we call grammar or vocabulary recovery, the degree to which a model was able to align its decision boundaries with the rules of the grammar that was used to simulate the data it was trained on. This is possible because for simulated data we not only know the ground truth label for each example, but also which positions are part of the background and thus contain no information about the class label, and which positions were altered by a grammar rule and thus do contain information about the class label. These position-level annotations (*background positions*, *grammar positions*) are provided for all simulated examples.

More formally, FIEs take a model $f(x)$, a target y and an example x_i of width n, and return z, an n-dimensional vector that contains the attribution value (also known as importance, relevance, contribution) of each input position to model $f(x)$ predicting target y. Please note that n is the sequence length of the example, not the number of features. For instance, if the input to the model is a 150 nt DNA sequence, x_i is a 150 by 4 matrix (one-hot encoded), containing 600 features, but its width $n = 150$. Feature attribution values in seqgra are grouped and reported at the position level, not the input feature level.

Attribution values are visualized with so-called grammar agreement plots, which are heatmaps depicting attributions and position-level annotations of several examples. The plots encode the attribution values in the color luminosity, where lighter colors indicate low values (low feature importance) and dark colors indicate high values (high feature importance). The position-level annotations are encoded in the color hue, with grammar positions in green and background positions in red.

2.4 Gradient-based feature importance evaluators

This large class of FIEs uses backpropagation to calculate the partial derivatives of the output, $f_y(x)$, with respect to the input, x_i. seqgra includes seven gradient-based FIEs off-the-shelf, whose implementations are based on code by Wang (2018).

The most basic FIE, *raw gradient* (Simonyan *et al.*, 2014), just returns the gradient with respect to the input example x_i

$$z_{\mathrm{RG}} = \frac{\partial f_y(x)}{\partial x_i}, \tag{1}$$

or short $\nabla f_y(x_i)$, where $f_j(\cdot)$ is the activation of the target neuron in the output layer, e.g. class j for multiclass classification tasks.

The absolute gradient method or *saliency* is defined as

$$z_{\mathrm{S}} = |\nabla f_y(x_i)|, \tag{2}$$

where $|x|$ applies the element-wise absolute value operation to vector x.

Gradient-x-input (Baehrens *et al.*, 2010; gradient times input) is defined as

$$z_{\mathrm{GI}} = x_i \nabla f_y(x_i). \tag{3}$$

Integrated Gradients (Sundararajan et al., 2017) take the average of multiple (here, $K = 100$) gradients evaluated along the linear path from the baseline x_0 (which in seqgra is the zero vector) to the input example x_i. The method is defined as

$$z_{\mathrm{IG}} = \frac{1}{K}\sum_{k}^{K} \nabla f_y\left(\frac{k}{K} x_i\right). \tag{4}$$

seqgra also supports gradient-based methods that alter the way the gradient is obtained using backpropagation, namely *Guided Backpropagation* (Springenberg *et al.*, 2015), *Deconvolution* (Zeiler and Fergus, 2014) and *DeepLIFT* (Shrikumar *et al.*, 2017). The details of these methods are beyond the scope of this work.

2.5 Model-agnostic feature importance evaluators

Model-agnostic FIEs do not require access to the gradients and make no assumptions about the structure of the model, hence the name. They rely solely on the ability to evaluate $f_y(x)$, for various altered versions of x.

Sufficient input subsets (SIS) (Carter *et al.*, 2019) is a perturbation-based method that identifies subsets of input features that are sufficient to keep $f_y(x) > \tau$, i.e. if all other features are masked, the class prediction does not change (is still above some threshold τ). Unlike gradient-based FIEs, which return a real-valued vector of feature attributions, SIS returns a binary vector, indicating for each feature whether it is part of an SIS or not.

2.6 Hardware infrastructure

Models presented in this paper were trained on three compute nodes with a total of six CPUs (2× Intel Xeon E5-2630 v4, 2× Intel Xeon Gold 6138, 2× Intel Xeon Gold 6240), 26 GPUs (8× NVIDIA GeForce GTX 1080 Ti with 11 GB GDDR5X, 10× NVIDIA GeForce RTX 2080 Ti with 11 GB GDDR6 and 8× NVIDIA Titan RTX with 24 GB GDDR6), and a total of 833 GB of main memory. The total GPU time (for training and evaluation) was roughly 12 GPU months.

2.7 Software infrastructure

All seqgra data presented in this paper was obtained on machines running Ubuntu 18.04.3 LTS, CUDA 10.1, cuDNN 7.6.5, Python 3.8, NumPy 1.19.2, TensorFlow 2.2.0, PyTorch 1.7.0 and R 4.0.

3 Results

3.1 seqgra provides a reproducible, simulation-based framework for neural network architecture evaluation

The method we describe in this paper (seqgra) generates synthetic biological sequence data according to predefined probabilistic rules in order to either (i) evaluate neural network architectures trained on these datasets or (ii) compare feature attribution methods in a setting with perfect dense (position-specific) labels. In the former scenario, the result would be a neural network architecture that—when trained on datasets generated from a similar set of rules—has high predictive performance and decision boundaries that closely reflect those set of generative rules. The goal of the latter approach is to investigate the interplay between grammar complexity and model complexity and how they influence feature attribution methods.

A dataset in the context of seqgra, whether obtained by simulation or experiment, is always divided into three subsets, training set, validation set and test set. Each of the subsets comprises a number of supervised examples, which are (x, y, a)-triplets. Here, the input variable x is a biological sequence (DNA, RNA, protein) of fixed or variable length, also referred to as sequence window or features; y is the target variable, the *condition* this example belongs to (e.g. cell type), which is either a mutually exclusive *class* or a non-mutually exclusive *label*, for multiclass classification tasks or multilabel classification tasks, respectively; and a is the positional annotation of the example, denoting for each position in x whether it is part of the *grammar* or part of the *background*. Grammar positions contain information related to y and are therefore important for classification, whereas background positions do not and are thus irrelevant for classification.

The core functionality of seqgra can be broken down into three components: (i) simulator, (ii) learner and (iii) evaluator. Each component corresponds to a distinct step in the pipeline depicted in Figure 1A.

In step 1, the simulator generates a synthetic dataset according to the specifications laid out in the data definition, a document that contains a precise description of the generated data, from the background nucleotide distribution to the set of probabilistic rules that determines how information about the condition y (label, class) is encoded in the sequence window x. This set of probabilistic rules is also referred to as *grammar* or sequence grammar throughout this manuscript (hence the name *seqgra*), and although related to formal grammars, seqgra's probabilistic rules are not expressed as and not equivalent to production rules in the context of formal language theory.

Schematic depictions of six toy datasets, generated from probabilistic rules of varying complexity, are shown in Figure 1B. In each case, the dataset contains examples belonging to one of four classes and the probabilistic rules determine how information about the class y (in this case, the cell type) is encoded in the sequence window x. The ability to recover this relationship during training is imperative for the model's predictive performance. The sequence windows of the examples are shown as gray bars with colored spots, where background positions are shown in gray and grammar positions are shown in color. In the first example, each of the four cell types can easily be identified by the presence of a class-specific k-mer at the center of the sequence window, a relationship that, unsurprisingly, can be learned perfectly (i.e. close to an ROC AUC of 1.0) and efficiently (i.e. with few training examples) by most neural network architectures. Since a set of rules as simple as the one used in example 1 will almost always be an inadequate description of any biological process, seqgra allows for various ways to increase the complexity. Example 2 represents a small step up in complexity by replacing the fixed, class-specific k-mer with a class-specific position weight matrix (PWM), which is a common representation of naturally occurring sequence elements, such as binding sites for a transcription factor. Another small step up in complexity is example 3, where the PWM is placed randomly within in sequence window. In example 4, none of the PWMs is class-specific, only a combination of PWMs. Rules like these could be used to model cell type specific chromatin accessibility that is dependent on the interaction between transcription factors. Examples 5 and 6 encode class information in the relative position of PWMs instead of their presence or absence, with example dataset 5 using class-specific order constraints and example dataset 6 class-specific spacing constraints.

Once the synthetic dataset is generated, it is used by the learner component in step 2 to train a neural network model. It is important to note that the learner only has access to x and y of the (x, y, a) example triplets, and the positional annotations a are only utilized in step 3. Analogous to the role of the data definition for the simulator in step 1, the model definition serves as a blueprint for the learner by providing a precise description of the neural network architecture, the loss function, the optimizer and hyperparameters of the training process, and thus ensuring a reproducible model creation, training and serving process for both PyTorch and TensorFlow models.

In step 3, the fully trained model from step 2 is then evaluated with the help of an array of conventional test set metrics and FIEs, such as Integrated Gradients (Sundararajan et al., 2017) and SIS (Carter *et al.*, 2019).

As a means to illustrate the various inputs and outputs of this pipeline, we prepared the results of a single seqgra analysis in Supplementary Figure S2 (using DNA sequences as input) and Supplementary Figure S3 (using protein sequences as input) and describe the process in Supplementary Section S1.7.

3.2 seqgra-enabled ablation analysis reveals most efficient neural network architecture

Ablation, a technique widely used in neuroscience to determine the functions of brain regions by removing them one by one, has been used similarly to identify the relevant components of an artificial neural network (Lillian *et al.*, 2018; Meyes *et al.*, 2019). We performed ablation analysis to determine the effects of dropout (Srivastava *et al.*, 2014) and batch normalization (Ioffe and Szegedy, 2015) on the predictive performance and grammar recovery of a basic neural network architecture with two hidden layers, a convolutional layer with 10 21-nt wide filters, followed by a dense layer with 5 hidden units, and dropout or batch normalization operations after each layer. Models were trained on binary classification datasets generated by grammars using class-specific HOMER motifs (see schematic in Fig. 2A), class-specific order of HOMER motifs (Fig. 2B) and class-specific spacing constraints between HOMER motifs (Fig. 2C). Test set precision–recall curve AUCs are shown for all models across all grammars in Figure 2D. Unsurprisingly, the predictive performance of all architectures increases with dataset size, and all architectures approach a PR AUC of 1.0 for sufficiently large datasets. But this analysis reveals a striking difference between the neural network architectures in terms of their efficiency, i.e. how many training examples are required to reach an AUC of approximately 1.0. On the grammars tested here, batch normalization had a

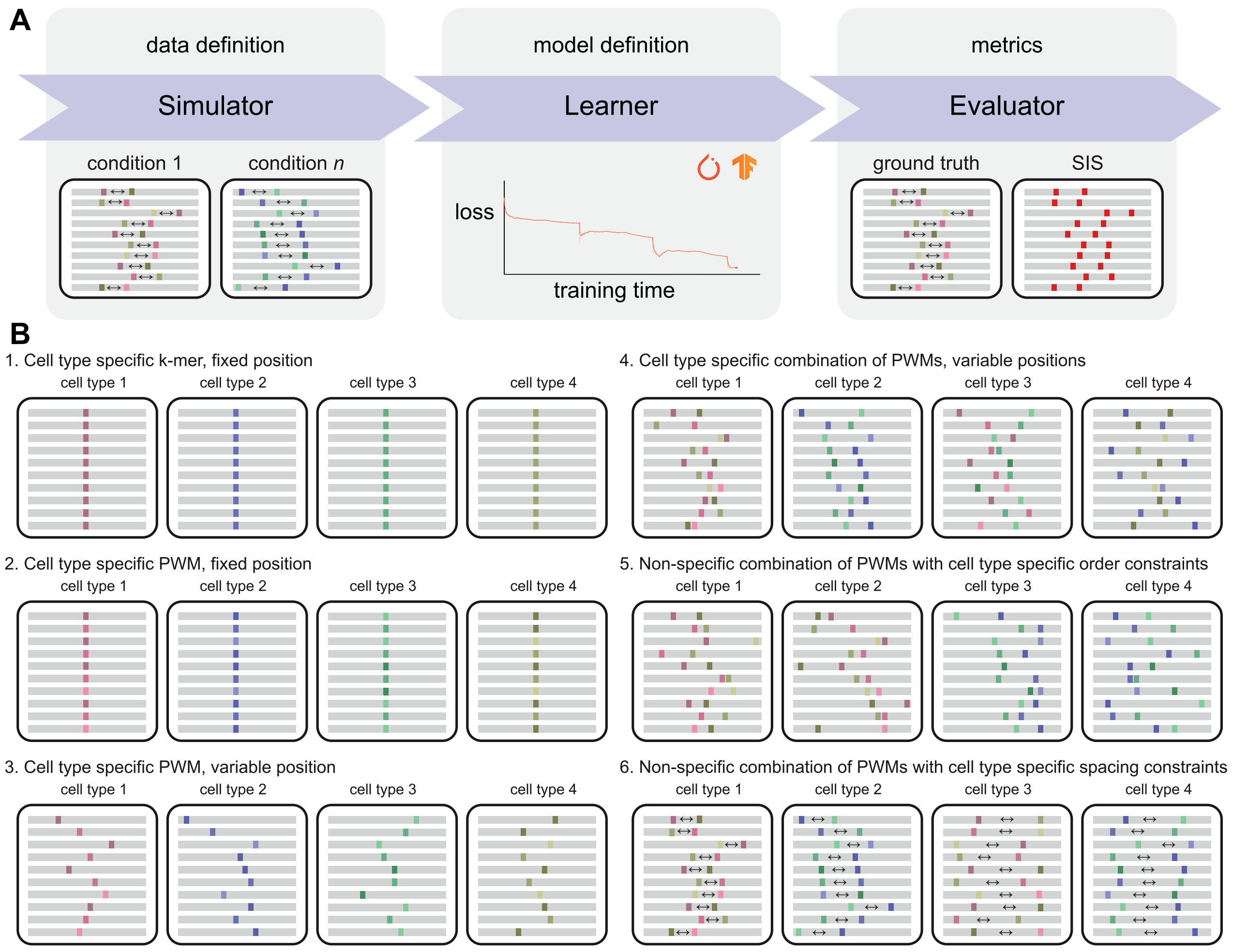
A
data definition
Simulator
condition 1
condition *n*
model definition
Learner
loss
training time
metrics
Evaluator
ground truth
SIS
B
1. Cell type specific k-mer, fixed position
2. Cell type specific PWM, fixed position
3. Cell type specific PWM, variable position
4. Cell type specific combination of PWMs, variable positions
5. Non-specific combination of PWMs with cell type specific order constraints
6. Non-specific combination of PWMs with cell type specific spacing constraints
cell type 1
cell type 2
cell type 3
cell type 4

Fig. 1. A framework for simulation-based evaluation of neural network architectures. (**A**) Schematic of the three main components: First, a simulator generates synthetic data according to the rules and specifications defined in the data definition file. Second, a learner creates a neural network model whose architecture and hyperparameters are specified in the model definition file, and trains it on the synthetic data from step 1. And third, the trained model is evaluated in terms of predictive performance and its ability to recover the rules specified in the data definition file. (**B**) The data definition specifies the basic properties of the synthetic data, including the alphabet (e.g. DNA, RNA, protein) and its distribution, as well as condition-specific rules (the *grammar*), which determine how information about the label *y* is encoded in the input *x*. (**C**) The model definition contains all information required to create and train the model. (**D**) A schematic of six simulated toy datasets for multiclass classification, where the classes *y* correspond to cell types and the input *x* are sequence windows (depicted as gray bars) that encode information about the class *y* at certain positions in *x* (colored areas). The rules that determine how this information is encoded range from basic (cell type specific *k*-mer at fixed position) to complex (non-specific combinations of PWMs with cell type specific spacing constraints)

negative effect on efficiency, requiring up to 100 000 examples more to converge than architectures without the operation. The architecture with dropout after each hidden layer was the most efficient and highest performing, both in terms of predictive performance and grammar recovery (i.e. the model's propensity to classify examples based on grammar positions) as shown in Figure 2E.

3.3 Deepsea dominates comparison of popular genomics deep learning architectures

Furthermore, we compared three popular neural network architectures used in the field of genomics, Basset (Kelley *et al.*, 2016), ChromDragoNN (Nair *et al.*, 2019) and DeepSEA (Zhou and Troyanskaya, 2015). All three architectures were devised with functional genomics datasets in mind and were originally trained on multilabel classification datasets obtained from numerous DNase-seq assays, with ChromDragoNN also utilizing RNA-seq and DeepSEA ChIP-seq data. With over 4 million (Basset), over 6 million (DeepSEA) and over 20 million (ChromDragoNN) trainable parameters, all three can be considered high-capacity models. The three architectures make use of commonly used building blocks such as convolutional, followed by dense layers (all three), max pooling and dropout operations (all three), ReLU activation functions (all three), batch normalization (Basset and ChromDragoNN) and skip connections (ChromDragoNN). Input and output layers were adjusted to fit the prediction task and architectures were trained on simulated datasets from scratch without pretraining on their original datasets.

We used the area under the microaveraged precision–recall curve to evaluate the test set predictive performance on four multiclass classification tasks (with 2, 10, 20 and 50 classes) and three or four grammars each, with a sequence window of 1000 nucleotides. The results are shown in Supplementary Figure S6A for binary classification, and Supplementary Figures S6B, S6C and S6D for multiclass classification with 10, 20 and 50 classes, respectively. The HOMER motifs used by the grammars presented here are listed in Supplementary Tables S2–S5. Each panel contains precision–recall AUCs of models trained on datasets generated by one grammar, using five different random seeds for simulation (error bars) and 19 different dataset sizes. The DeepSEA architecture exhibited an at times substantially higher predictive performance than Basset and ChromDragoNN and was the highest performing architecture on all tested datasets. While DeepSEA is the preferred architecture on datasets derived from the grammars we tested, this is not necessarily true for datasets with other grammars or experimentally obtained

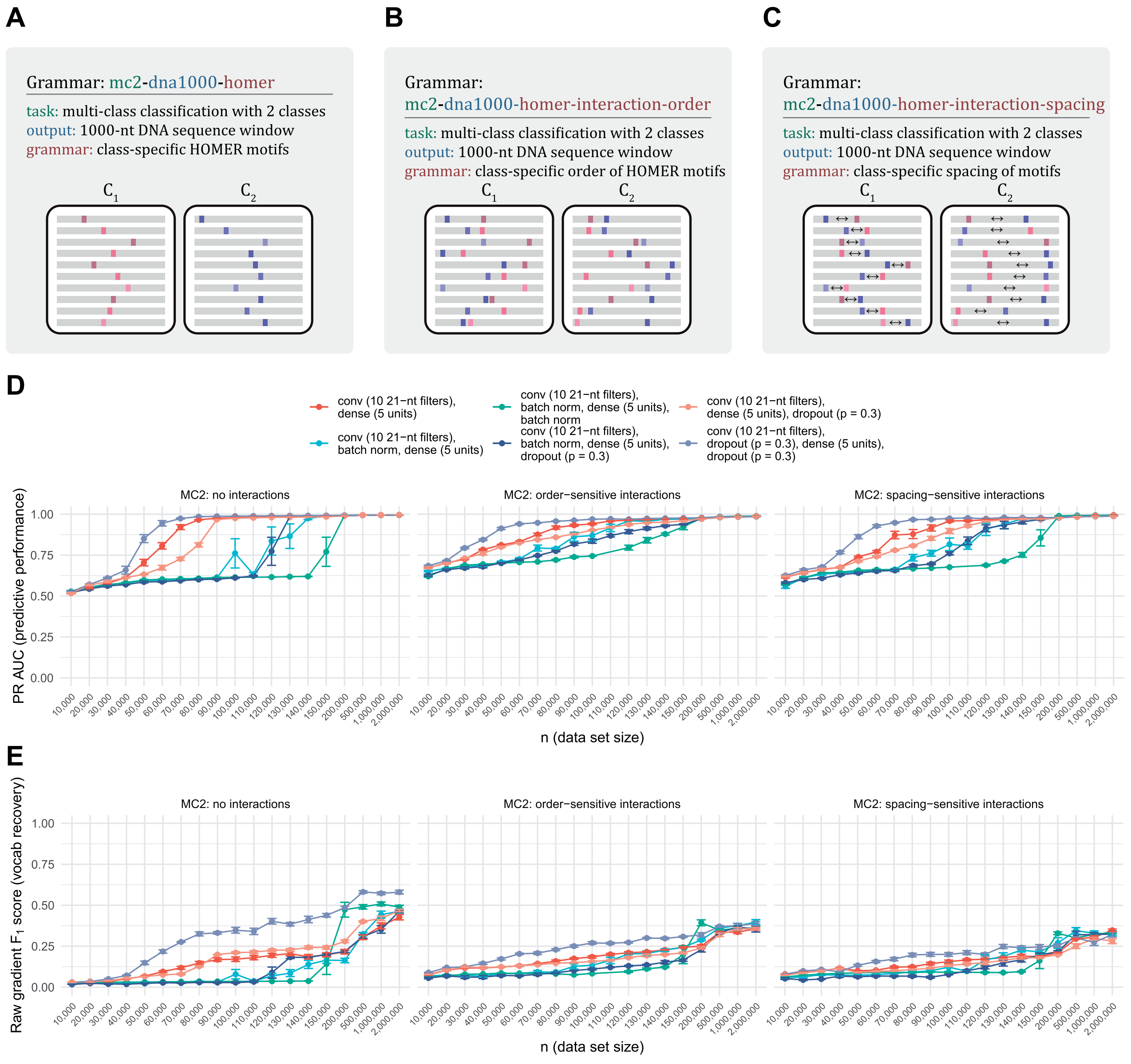

Fig. 2. seqgra-enabled ablation analysis reveals most efficient neural network architecture. (**A**) Schematic of binary classification grammar using class-specific HOMER motifs as sequence elements. (**B**) Schematic of binary classification grammar using class-specific order of HOMER motifs. (**C**) Schematic of binary classification grammar using class-specific spacing of HOMER motifs. (**D**) Predictive performance of six neural network architectures with and without batch normalization and dropout. (**E**) Vocabulary recovery of six neural network architectures with and without batch normalization and dropout

data. ChromDragoNN, e.g. is intended to be also trained on RNA-seq data, which we did not provide. Interestingly, we observed that high-capacity architectures such as those tested here perform better on datasets generated by grammars that include interactions, specifically interactions that encode the class label in the order or spacing of the interacting sequence elements. This is not the case for small-scale architectures with less than 100 000 trainable parameters, which, as expected, do better on grammars without interactions, where the class label is encoded in the presence of class-specific sequence elements.

3.4 High predictive performance of simulation-vetted neural network architecture recapitulated with ChIP-seq data

In this section, we address the question of whether neural network architectures that perform well on simulated data also succeed on data obtained experimentally. We decided to model the well-known hetero-dimeric pair of transcription factors SOX2 and POU5F1, whose spacing constraints were previously characterized (Chew *et al.*, 2005; Guo *et al.*, 2012). To that end, we used the HOMER motifs SOX2_HUMAN.H11MO.0.A and PO5F1_HUMAN.H11MO.1.A as sequence elements in the data definition. We also included spacing constraints (0–3 bp between SOX2 and PO5F1 motifs). Figure 3A shows a schematic depiction of the analysis.

The experimental dataset was based on two ChIP-seq assays, which targeted the two transcription factors. The preprocessed data were obtained from the Cistrome Data Browser (Mei *et al.*, 2017), specifically the data associated with GEO IDs GSM1701825 for SOX2 and GSM1705258 for POU5F1.

We evaluated the same neural network architectures on both the simulated and the experimental datasets. The architecture described in Figure 3B with one fully connected layer (not counting the output layer) is an example of an architecture that does not assume any

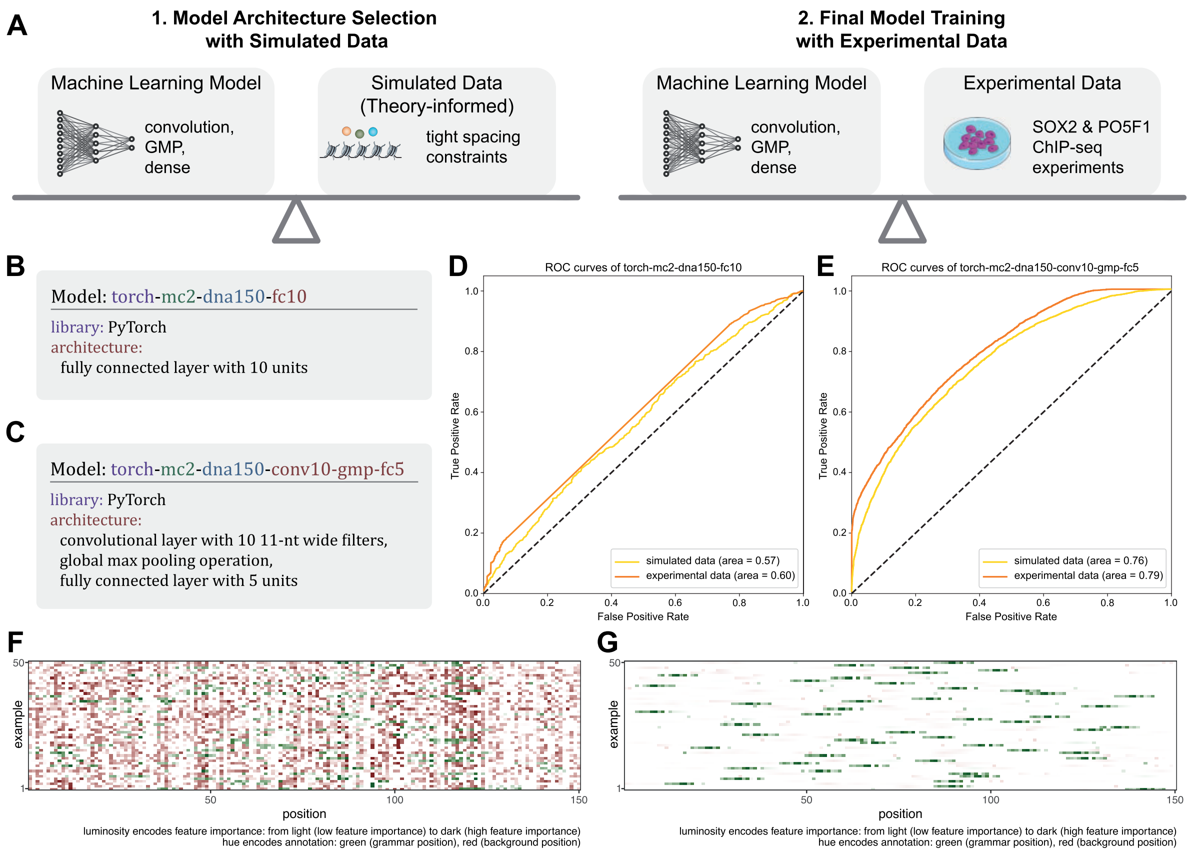

Fig. 3. Predictive performance and grammar recovery of various model architectures on simulated and experimental data. (**A**) Schematic of model selection process: first, identify suitable model architectures on simulated data; second, train models with simulation-vetted architectures on experimental data. (**B**) Naïve neural network architecture with fully connected layer. (**C**) Grammar-informed neural network architecture with convolutional layer, global max pooling and fully connected layer. (**D**) Predictive performance of naive architecture, trained and evaluated on simulated and experimental data. (**E**) Predictive performance of grammar-informed architecture, trained and evaluated on simulated and experimental data. (**F**) Grammar agreement plot (Integrated Gradients) of naive architecture, trained on experimental data. (**G**) Grammar agreement plot (Integrated Gradients) of grammar-informed architecture, trained on experimental data

structure in the input. It is a naive architecture in the sense that it was constructed without any knowledge about the grammar that was used to simulate the data. The architecture described in Figure 3C, on the contrary, makes assumptions about the data that are in agreement with the grammar, such as a 1D spatial structure with information encoded in 11-nt long code words (enough to cover the SOX2-POU5F1 interaction), whose position in the sequence window is irrelevant.

As expected, the test set predictive performance of the naive architecture (Fig. 3D) was significantly lower than the grammar-informed architecture (Fig. 3E). Furthermore, the performance on the simulated data proved to be a good predictor for the performance on the experimental data (Fig. 3D and E).

The agreement between feature importance and the grammar positions, a proxy for a model's ability to recover the SOX2 and POU5F1 motifs, is shown in Figure 3F for the naive architecture and in Figure 3G for the grammar-informed architecture. The grammar-informed model's predictions were based almost exclusively on grammar positions (positions that contained SOX2 and POU5F1 motifs), whereas this was not the case for the naive model. Both panels were created with the Integrated Gradients FIE.

4 Discussion

In this paper, we introduced seqgra, a deep learning infrastructure method for genomics. It is intended to streamline the development of deep learning models for biological sequence-based prediction tasks, by providing a reproducible unified framework for (i) flexible, rule-based synthetic data generation; (ii) model training and (iii) model evaluation with conventional test set metrics and feature attribution methods. This three-step pipeline supports datasets obtained by simulation and experiment, models implemented in PyTorch and TensorFlow, and numerous gradient-based feature attribution methods as well as SIS, a model-agnostic feature attribution method, in addition to conventional ROC and precision–recall curves for model evaluation. Our method greatly simplifies an array of commonly performed diagnostics and performance assessments of deep learning models, such as ablation analysis, estimated dataset size requirements and tolerated noise thresholds. The simulator and the language of the probabilistic rules are flexible enough to span multiclass and multilabel classification tasks with any number of classes or labels, DNA or amino acid sequence windows of variable or fixed length, class-dependent background distributions, sequence elements defined as PWMs or list of *k*-mers with associated probabilities, and interactions between sequence elements with associated order or spacing constraints.

Moreover, the controlled environment of data simulation and reproducible model training, serving and evaluation makes seqgra a suitable testbed for feature attribution and interpretability methods and their interdependencies with neural network architectures and the complexity level of the training data. Moreover, the framework can be used to perform extensive comparisons between deep learning libraries, which are rarely done (see Supplementary Figs S10 and S11) or identify undocumented behavior of the deep learning technology stack, such as an unusual training instability caused by a random seed of zero on some grammar-architecture combinations, which is reproducible and occurs in both PyTorch and TensorFlow (see Supplementary Figs S7 and S8).

To avoid confusion, we would like to point out that seqgra is not a neural architecture search technique in the sense that it will not propose suitable neural network architectures for a particular dataset. The model definition is an input, not an output of the seqgra pipeline. However, seqgra can be used in conjunction with neural architecture search, such as AMBER (Zhang *et al.*, 2021), a neural architecture search method for architectures aimed at genomics prediction tasks, or general hyperparameter optimization methods, such as Hyperband (Li *et al.*, 2016). Likewise, seqgra currently does not automatically explore the space of generative rules to find a set of rules that match a particular experimental situation. The rules underlying the generative process of the simulator are an input to seqgra (the data definition) and usually based on domain expert. Furthermore, if the goal is to find the model with the highest predictive performance on a particular experimental dataset, a general hyperparameter optimization approach such as Hyperband or NAS when exploring a carefully selected hyperparameter subspace, is expected to outperform seqgra. However, these (in all likelihood) very-high-capacity models tend to be less useful when the primary concern is not predictive performance, but a better understanding of the underlying rules. Oftentimes a simpler model with lower predictive performance is better than a complex model with higher predictive performance, especially when dealing with biological data where noise levels are high and often systematic, e.g. biases introduced by the assay that are present in both training and test sets, but are not part of the underlying biological systems. The predictive performance gains that incorporate these are often undesired.

One caveat of all simulation-based approaches is the inevitable gap between simulated and real-world datasets, in the sense that the former is always a simplified approximation of the latter. Thus, insights gained from simulated data might not carry over to the experimental world. In fact, to a certain degree, this will always be the case. However, while high-performing neural network architectures on simulated data might not perform as highly on experimental data, the opposite is rarely the case, i.e. low-performing architectures in simulation are unlikely to improve when trained on noisier and/or smaller experimental datasets. Moreover, if a model performs well on both simulated and experimental data, that does not imply that the underlying grammar rules of the simulated data are similar to the rules governing the experiment. The opposite situation, where the model performs well on simulated data and poorly on experimental data, in contrast, is oftentimes more insightful as it suggests that either the underlying rules are different or if the rules are similar, the model fails to learn them because of high noise levels in the experimental data or a paucity of experimental data available for training.

While the intricacies of noisy and biased high-throughput genomics experiments make for highly complex and poorly understood datasets, training highly complex alchemy-like (Hutson, 2018a) deep neural networks on them contributes little to a mechanistic understanding of the biological processes that are at work underneath and might worsen the reproducibility crisis in both machine learning (Hutson, 2018b) and biology (Baker, 2016; Begley and Ellis, 2012). Simulated data, however, are perfectly understood, its noise levels controlled and any biases artificially introduced and accounted for, which makes it an excellent environment for model evaluation. With seqgra, the clean room of simulated data and a precise description of the patterns in the data (i.e. the probabilistic rules in the data definition) on the one end is paired with an array of feature attribution methods on the other, to answer questions that are often impossible to answer with poorly understood genomics data. One such question is whether the predictions of the model are based on those parts of the input that are in fact relevant for the phenomenon that is predicted, or, to put it another way, whether the model was able to recover the underlying rules of the dataset.

Funding

We gratefully acknowledge funding from National Institutes of Health (NIH) [1R01HG008754 to D.K.G. and 1R01NS109217 to D.K.G.]; and National Science Foundation Graduate Research Fellowship [1122374 to J.H.].

Conflict of Interest: none declared.

Data availability

The source code of the seqgra package is hosted on GitHub (https://github.com/gifford-lab/seqgra) and licensed under the MIT license. seqgra is part of the Python Package Index PyPI and can be installed using pip, the Python package installer. Extensive documentation can be found at https://kkrismer.github.io/seqgra.

The data underlying this article will be shared on reasonable request to the corresponding author.

References

Avsec,Z. *et al.* (2019) The Kipoi repository accelerates community exchange and reuse of predictive models for genomics. *Nat. Biotechnol.*, **37**, 592–600.

Baehrens,D. *et al.* (2010) How to explain individual classification decisions. *J. Mach. Learn. Res.*, **11**, 1803–1831.

Baker,M. (2016) 1,500 scientists lift the lid on reproducibility. *Nature*, **533**, 452–454.

Begley,C.G. and Ellis,L.M. (2012) Drug development: raise standards for preclinical cancer research. *Nature*, **483**, 531–533.

Boyle,A.P. *et al.* (2008) High-resolution mapping and characterization of open chromatin across the genome. *Cell*, **132**, 311–322.

Buenrostro,J.D. *et al.* (2013) Transposition of native chromatin for fast and sensitive epigenomic profiling of open chromatin, DNA-binding proteins and nucleosome position. *Nat. Methods*, **10**, 1213–1218.

Carter,B. *et al.* (2019) What made you do this? understanding black-box decisions with sufficient input subsets. *Proc. Mach. Learn. Res.*, **89**, 567–576.

Chen,K.M. *et al.* (2019) Selene: a PyTorch-based deep learning library for sequence data. *Nat. Methods*, **16**, 315–318.

Chew,J.L. *et al.* (2005) Reciprocal transcriptional regulation of Pou5f1 and Sox2 via the Oct4/Sox2 complex in embryonic stem cells. *Mol. Cell. Biol.*, **25**, 6031–6046.

Guo,Y. *et al.* (2012) High resolution genome wide binding event finding and motif discovery reveals transcription factor spatial binding constraints. *PLoS Comput. Biol.*, **8**, e1002638.

Heinz,S. *et al.* (2010) Simple combinations of lineage-determining transcription factors prime cis-regulatory elements required for macrophage and B cell identities. *Mol. Cell*, **38**, 576–589.

Hutson,M. (2018a) Ai researchers allege that machine learning is alchemy. *Science*. https://doi.org/10.1126/science.aau0577.

Hutson,M. (2018b) Artificial intelligence faces reproducibility crisis. *Science*, **359**, 725–726.

Ioffe,S. and Szegedy,C. (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. In: *Proceedings of the 32nd International Conference on Machine Learning (ICML'15), Lille, France*, Vol. **37** of *Proceedings of Machine Learning Research*, pp. 448–456.

Kelley,D.R. *et al.* (2016) Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks. *Genome Res.*, **26**, 990–999.

Li,L. *et al.* (2016) Efficient hyperparameter optimization and infinitely many armed bandits. *CoRR*, abs/1603.06560.

Lillian,P. *et al.* (2018) Ablation of a robot's brain: neural networks under a knife. *CoRR*, abs/1812.05687.

Mei,S. *et al.* (2017) Cistrome Data Browser: a data portal for ChIP-Seq and chromatin accessibility data in human and mouse. *Nucleic Acids Res.*, **45**, D658–D662.

Meyes,R. *et al.* (2019) Ablation studies in artificial neural networks. *CoRR*, abs/1901.08644.

Nair,S. *et al.* (2019) Integrating regulatory DNA sequence and gene expression to predict genome-wide chromatin accessibility across cellular contexts. *Bioinformatics*, **35**, i108–i116.

Piovesan,A. *et al.* (2019) On the length, weight and GC content of the human genome. *BMC Res. Notes*, **12**, 106.

Quang,D. and Xie,X. (2016) DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences. *Nucleic Acids Res.*, **44**, e107.

Shrikumar,A. *et al.* (2017) Learning important features through propagating activation differences. *Proc. Mach. Learn. Res.*, **70**, 3145–3153.

Simonyan,K. *et al.* (2014) Deep inside convolutional networks: visualising image classification models and saliency maps. In: Bengio Y. and LeCun Y. (eds.), *2nd International Conference on Learning Representations (ICLR 2014), Banff, AB, Canada, April 14–16, 2014, Workshop Track Proceedings*.

Springenberg,J.T. *et al.* (2015) Striving for simplicity: the all convolutional net. In: Bengio Y. and LeCun Y. (eds.), 3rd *International Conference on Learning Representations (ICLR 2015), San Diego, CA, USA, May 7–9, 2015, Workshop Track Proceedings*.

Srivastava,N. *et al.* (2014) Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, **15**, 1929–1958.

Sundararajan,M. *et al.* (2017) Axiomatic attribution for deep networks. In: Precup D. and Naïve Y.W. (eds.), *Proceedings of the 34th International Conference on Machine Learning (ICML 2017), Sydney, NSW, Australia, 6–11 August 2017*, Vol. 70. Proceedings of Machine Learning Research, pp. 3319–3328.

Wang,Y. (2018) *PyTorch-Visual-Attribution*. https://github.com/yulongwang12/visual-attribution (22 February 2022, date last accessed).

Zeiler,M.D. and Fergus,R. (2014) Visualizing and understanding convolutional networks. In Fleet D.J. *et al.* (eds.), Computer Visi–n-ECCV 20–4-13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part I, Vol. 8689 of Lecture Notes in Computer Science. Springer, pp. 818–833.

Zhang,Z. *et al.* (2021) An automated framework for efficiently designing deep convolutional neural networks in genomics. *Nat. Mach. Intell.*, **3**, 392–400.

Zhou,J. and Troyanskaya,O.G. (2015) Predicting effects of noncoding variants with deep learning-based sequence model. *Nat. Methods*, **12**, 931–934.