Janggu - Deep learning for genomics

Wolfgang Kopp[1,*], Remo Monti[1,2], Annalaura Tamburrini[1,3], Uwe Ohler[1,4], Altuna Akalin[1,*]

1 Berlin Institute for Systems Biology, Max Delbrueck Center for Molecular Medicine, 10115 Berlin, Germany.
2 Digital Health Center, Hasso Plattner Institute, University of Potsdam, 14482 Potsdam, Germany.
3 Department of Biology, Centro di Bioinformatica Molecolare, University of Rome "Tor Vergata", 00133 Rome, Italy.
4 Department of Biology, Humboldt University, 10115 Berlin, Germany.
∗ Correspondence to: wolfgang.kopp@mdc-berlin.de, altuna.akalin@mdc-berlin.de

Abstract

Motivation In recent years, numerous applications have demonstrated the potential of deep learning for an improved understanding of biological processes. However, most deep learning tools developed so far are designed to address a specific question on a fixed dataset and/or by a fixed model architecture. Adapting these models to integrate new datasets or to address different hypotheses can lead to considerable software engineering effort. To address this aspect we have built *Janggu*, a python library that facilitates deep learning for genomics applications.
Results Janggu aims to ease data acquisition and model evaluation in multiple ways. Among its key features are special dataset objects, which form a unified and flexible data acquisition and pre-processing framework for genomics data that enables streamlining of future research applications through reusable components. Through a numpy-like interface, the dataset objects are directly compatible with popular deep learning libraries, including keras. Furthermore, *Janggu* offers the possibility to visualize predictions as genomic tracks or by exporting them to the BIGWIG format. We illustrate the functionality of *Janggu* on several deep learning genomics applications. First, we evaluate different model topologies for the task of predicting binding sites

for the transcription factor JunD. Second, we demonstrate the framework on published models for predicting chromatin effects. Third, we show that promoter usage measured by CAGE can be predicted using DNase hypersensitivity, histone modifications and DNA sequence features. We improve the performance of these models due to a novel feature in *Janggu* that allows us to include high-order sequence features. We believe that *Janggu* will help to significantly reduce repetitive programming overhead for deep learning applications in genomics, while at the same time enabling computational biologists to assess biological hypotheses more rapidly.
Availability *Janggu* is freely available under a GPL-v3 license on
https://github.com/BIMSBbioinfo/janggu or via
https://pypi.org/project/janggu

Keywords: Deep learning, Machine Learning, Genomics, Epigenomics

Background

The recent explosive growth of biological data, particularly in the field of regulatory genomics, has continuously improved our understanding about regulatory mechanism in cell biology [1]. Meanwhile, the remarkable success of deep neural networks in other areas, including computer vision, has attracted attention in computational biology as well. Deep learning methods are particularly attractive in this case, as they promise to extract knowledge in a data-driven fashion from large datasets while requiring limited domain expertise [2]. Since their introduction [3, 4], deep learning methods have dominated computational modeling strategies in genomics where they are now routinely used to address a variety of questions ranging from the understanding of protein binding from DNA sequences [3], epigenetic modifications [4, 5, 6], predicting gene-expression from epigenetic marks [7], or predicting the methylation state of single cells [8].

Depsite the success of these numerous deep learning solutions and tools, their broad adaptation by the bioinformatics community has been limited. This is partially due to the low flexibility of the published methods to adapt to new data, which often requires a considerable engineering effort. This situation illustrates a need for software frameworks that allow for a fast turnover when it comes to addressing new hypotheses, integrating new datasets, or experimenting with new neural network architectures.

In fact, several recent packages, including pysster [9], kipoi [10] and selene

[11], have been proposed to tackle this issue on different levels. However, they are limited in their expressiveness due to the restricted programming interface [9], a focus on reproducibility and reusability of trained models but not the entire training process, [10] or the adoption of a specific neural network library through a tight integration [11]. All of them have in common that the support of different data types beyond sequence is limited.

To address some of these shortcomings, we have developed a novel python library for deep learning in genomics, called *Janggu*. The library is identically named to a Korean percussion instrument that is shaped like an hourglass and whose two ends reflect the two ends of a deep learning application, namely data acquisition and evaluation. The library supports flexible prototyping of neural network models by separating the pre-processing and dataset specification from the modelling part. Accordingly, *Janggu* uses dedicated genomics dataset objects. These objects provide easy access and pre-processing capabilities to fetch data from common file formats, including FASTA, BAM, BIGWIG and BED files (see Fig. 1), and they are directly compatible with commonly used deep learning libraries, such as keras or scikit-learn. In this way, they effectively bridge the gap between commonly used file formats in genomics and the python data format that is understood by the deep learning libraries. The dataset objects can be easily reused for different applications, and they place no restriction on the model architecture to be used with. A key advantage of establishing reusable and well-tested dataset components is to allow for a faster turnaround when it comes to setting up deep learning models and increased flexibility for addressing a range of questions in genomics. As a consequence, we expect significant reductions in repetitive software engineering aspects that are usually associated with the pre-processing steps.

We illustrate *Janggu* on three use cases: 1) predicting transcription factor binding of JunD, 2) using and improving published deep learning architectures, and 3) predicting normalized CAGE-tags counts at promoters. In these examples, different data formats are consumed, including FASTA, BIGWIG, BAM and narrowPeak files. Here, we also make use of *Janggu*'s ability of using higher-order sequence features (see Hallmarks), and show that this leads to significant performance improvements.

Results

Hallmarks of Janggu

Janggu offers two special dataset classes: *Bioseq* and *Cover*, which can be used to conveniently load genomics data from a range of common file formats, including FASTA, BAM, BIGWIG or BED files. Biological sequences (e.g. from the reference genome) and coverage information (e.g. from BAM, BIGWIG or BED files) are loaded for user-specified regions of interest (ROI), which are provided in BED-like format. Since *Bioseq* and *Cover* both mimic a minimal numpy interface, the objects may be directly consumed using e.g. keras or scikit-learn.

Bioseq and *Cover* provide a range of options, including the binsize, step size, or flanking regions for traversing the ROI. The data may be stored in different ways, including as ordinary numpy arrays, as sparse arrays or in hdf5 format, which allow the user to balance the trade-off between speed and memory footprint of the application. A built-in caching mechanism helps to save processing time by reusing previously generated datasets. This mechanism automatically detects if the data has changed and needs to be reloaded.

Furthermore, *Cover* and *Bioseq* expose dataset-specific options. For instance, coverage tracks can be loaded at different resolution (e.g. base-pair or 50-bp resolution) or be subjected to various normalization and transformation steps, including TPM-normalization. *Bioseq* also enables the user to work with both DNA and protein sequences. Here, sequences can be one-hot encoded using higher-order sequence features, allowing the models to learn e.g. di- or tri-mer based motifs.

Finally, *Janggu* offers a number of model evaluation and interpretation features: 1) Commonly used performance metrics can be directly used within the framework, including the area under the receiver operator characteristic curve (auROC) or the area under the precision-recall curve (auPRC). 2) Predictions obtained for any deep learning library usually take the form of numpy arrays. These arrays can be converted back to a coverage object, which eventually ensures that the user does not have to maintain correspondences between two sets of indices, namely the numpy-array indices and genomic intervals. Hence, the application can be phrased more naturally in terms of genomic coordinates, and numpy-array indices are abstracted away by *Janggu*. 3) Conversion to coverage objects also offers the possibility to export predictions to BIGWIG format or to inspect the results directly via *Janggu*'s built-in *plotGenomeTrack* function. 4) Input feature importance can be investigated using the integrated gradients method [12] and 5) changes of the prediction score can studied for single nucleotide variants taking advantage of the higher-order sequence representation. A schematic overview is illustrated in Fig. 1. Further details on its functionality are available in the documentation at https://janggu.readthedocs.io.

Prediction of JunD binding

To showcase different *Janggu* functionalities, we defined three example problems to solve entirely within the framework. We start by predicting the binding events of the transcription factor JunD. JunD binding sites exhibit strong interdependence between nucleotide positions [13], suggesting that it might be beneficial to take the higher-order sequence composition directly into account. To this end, we introduce a higher-order one-hot encoding of the DNA sequence that captures e.g. di- or tri-nucleotide based motifs. For example, for a sequence of length N, the di-nucleotide one-hot encoding corresponds to a $16 \times N - 1$ matrix, where each column contains a single *one* in the row that is associated with the di-nucleotide at that position. We shall refer to mono-, di- and tri-nucleotide encoding as order one, two and three, respectively. In contrast to mono-nucleotide input features, higher-order features directly capture correlations between neighboring nucleotides.

For JunD target predictions, we observe a significant improvement in area under the precision recall curve (auPRC) on the test set when using the higher order sequence encoding compared to the mono-nucleotide encoding (see Fig. 2A, red). The median performance gain across five runs amounts to $\Delta auPRC = 8.3\%$ between order 2 and 1, as well as $\Delta auPRC = 9.3\%$ between order 3 and 1.

While the use of higher-order sequence features uncovers useful information for interpreting the human genome, the larger input and parameter space might make the model prone to overfitting, depending on the amount of data and the model complexity. We tested whether dropout on the input layer, which randomly sets a subset of ones in the one-hot encoding to zeros, would improve model generalization [14]. Using dropout on the input layer should also largely preserve the information content of the sequence encoding, as the representation of higher orders is inherently redundant due to overlapping neighboring bases.

In line with our expectations, dropout leads to a slight further performance improvement for tri-nucleotide-based sequence encoding. On the other hand, for mono-nucleotide-based encoding we observe a performance decrease. We observe slightly worse performance also when using di-nucleotide-based encoding, suggesting that the model is over-regularized with the addition of dropout. However, dropout might still be a relevant option for the di-nucleotide based encoding if the amount of data is relatively limited (see Fig. 2A).

As many other transcription factors, JunD sites are predominately localized in accessible regions in the genome, for instance as assayed via DNase-seq [15]. To investigate this further, we set out to predict JunD binding from the raw DNase cleavage coverage profile in 50 bp resolution extracted from BAM files of two independent replicates simultaneously (from ENCODE and ROADMAP, see Methods).

Raw read coverage obtained from BAM files is inherently biased, e.g. due to differences in sequencing depths etc., which requires normalization in order to achieve comparability between experiments. As a complementary approach, data augmentation has been shown to improve generalization of neural networks by increasing the amount of data by additional perturbed examples of the original data points [16]. Accordingly, we compare TPM normalization and Z score normalization of $log(count + 1)$ in combination with data augmentation by flipping the 5' to 3' orientation of the coverage tracks. To test the effectiveness of normalization and data augmentation, we swapped the input DNase experiments from ENCODE and ROADMAP between training and test phase. The more adequate the normalization, the higher we anticipate the performance to be on the test set.

We find that both TPM and Z score after $log(count + 1)$ transformation lead to improved performance compared to applying no normalization, with the Z score after $log(count + 1)$ transformation yielding the best results (see Fig. 2B). The additional application of data augmentation tends to slightly improve the performance for predicting JunD binding from DNase-seq (see Fig. 2B).

Next, we build a combined model for predicting JunD binding based on the DNA sequence and DNase coverage tracks. To that end, we used the same initial layers as for the order-3 DNA model and the DNase-specific models using Z score after $log(count+1)$-normalization with orientation flipping. We removed their output layers, concatenated the top most hidden layers, and added a new sigmoid output layer. We trained the joint model from scratch using randomly initialized weights for all layers and found that its performance significantly exceeded the performance of the individual DNA and DNase submodels, indicating that both ingredients contributed substantially to the predictive performance (compare Fig. 2A-C).

As a means to inspect the plausibility of the results apart from summary performance metrics (e.g. auPRC), *Janggu* features a built-in genome track plotting functionality that can be used to visualize the agreement between predicted and known binding sites, or the relationship between the predictions and the input coverage signal for a selected region (Fig. 2D). Input importance attribution using integrated gradients [12] additionally highlights the relevance of sequence features for the prediction, which in the case of the JunD prediction task reveals sequence patches reminiscent of the known JunD binding motif (with the canonical sequence motif TGACTCA) close to the center of the predicted peak (Fig. 2D).

Predicting chromatin profiles from genomic sequences

Predicting the function of non-coding sequences in the genome remains a challenge. In order to address this challenge and assess the functional relevance of non-coding sequences and sequence variants, multiple deep learning based models have been proposed. These models learn the genomic sequence features that give rise to chromatin profiles such as transcription binding sites, histone modification signals or DNase hypersensitive sites. We adopted two published neural network models that are designed for this purpose, which have been termed DeepSEA and DanQ [4, 17]. We rebuilt these models using the *Janggu* framework to predict the presence (or absence) of 919 genomic and epigenetic features, including DNase hypersensitive sites, transcription factor binding events and histone modification marks, from the genomic DNA sequence. To that end, we gathered and reprocessed the same features, making use of *Janggu*'s pre-processing functionality [4]. Both published models were adapted to scan both DNA strands simultaneously in the first layer rather than just the forward strand as this leads to slight performance improvements (see DnaConv2D layer, *Janggu* documentation). Then we assessed the performance of the different models by considering different context window sizes (500bp, 1000bp and 2000bp) as well as different one-hot encoding representations (based on mono-, di- and tri-nucleotide content).

First, as reported previously [17], we confirm that the DanQ model consistently outperforms the DeepSEA model regardless of the context window size, one-hot encoding representation and features type (e.g. histone modification, DNase hypersensitive sites and TF binding sites) (see Fig. S.1). Second, in line with previous reports [6, 4], we find the performance for histone modifications and histone modifiers (e.g. Ezh2, Suz12, etc.) to benefit from extending the context window sizes (see Fig. 3A and S.1) By contrast, elongating the context window yields similar performance for accessible sites and transcription factor binding-related features.

Third, higher-order sequence encoding influences predictions for histone modification, DNase and TF binding associated features differently. For histone modification predictions we observe similar performance for higher-order and mono-nucleotide based one-hot encoding higher-order and mono-nucleotide based one-hot encoding (see Fig. 3B,C). For the DNase accessibility we observe slight but consistent improvements. Half of the DNase associated features exhibit at least a 4% auPRC increase between the mono- and tri-nucleotide representation (see Fig. 3D). Finally, for the majority of transcription factor binding predictions we find mild or substantial improvements (see Fig. 3E). Among the most prominent performance improvements we obtain Nrsf, Pol3, Sp2, etc. (see Fig. 3B).

Predicting CAGE-signal at promoters

Finally, we used *Janggu* for the prediction of promoter usage of protein coding genes. Specifically, we built a regression application for predicting the normalized CAGE-tag counts at promoters of protein coding genes based on chromatin features (DNase hypersensitivity and H3K4me3 signal) and/or DNA sequence features. Due to the limited amount of data for this task, we pursue a per-chromosome cross-validation strategy (see Methods).

We trained a model using only the DNA sequence as input with different one-hot encoding orders. Consistent with our JunD prediction analysis, we observe that the use of higher-order sequence features markedly improves the average Pearson's correlation from 0.533 to 0.559 and 0.585 for mono-nucleotide features compared to di- and tri-nucleotide based features, respectively (see Tab. 1). Predictions from chromatin features alone yield a substantially higher average Pearson's correlation of 0.777 compared to using the DNA sequence models (see Tab. 1).

Similar to the previous sections, we concatenate the individual top most hidden layers and add new output layer to form a joint DNA and chromatin model. Consistent with our results from the JunD prediction, the Pearson's correlation between observed and predicted values increases for the combined model (see Tab. 1 and Fig. 4), even though the difference seems to be subtle in this scenario. The results also show that chromatin features vastly dominate the prediction accuracy compared to the contribution of the DNA sequence. This is expected due to the fact that the DNA sequence features are collected only from a narrow window around the promoter. On the other hand, the chromatin features reflect activation not only due to the local context, but also due to indirect activation from distal regulatory elements, e.g. enhancers.

Discussion

We present a novel python library, called *Janggu*, that facilitates deep learning in genomics. The library includes dataset objects that manage the extraction and transformation of coverage information as well as fetching biological sequence directly from a range of commonly used file types, including FASTA, BAM or BIGWIG. These dataset objects may be consumed directly with numpy-compatible deep learning libraries, e.g. keras, due to the fact that they mimic a minimal numpy interface, which in turn reduces the software engineering effort concerning the data acquisition for a range of deep learning applications in genomics. *Janggu* additionally facilitates utilities to monitor the training, performance evaluation and interpretation. For example, model prediction or features can be inspected using a built-in genome browser or they may be exported to BIGWIG files for further investigation. Input feature importance can be analyzed using integrated gradient and variant effects may assessed for a given VCF format file.

We have demonstrated the use of *Janggu* for three case studies 1) that required different file formats (FASTA, BAM, BIGWIG, BED and GFF), 2) different pre-processing and data augmentation strategies, 3) that demonstrated the advantage of one-hot encoding of higher-order sequence features (representing mono-, di-, and tri-nucleotide sequences), 4) for a classification and regression task (JunD prediction and published models) and a regression task (CAGE-signal prediction) and utilizing per-chromosome cross-validation. This illustrates our tool is readily applicable and flexible to address a range of questions allowing users to more effectively concentrate on testing biological hypothesis.

Throughout the use cases we confirmed that higher-order sequence features improve deep learning models, because they simultaneously convey information about the DNA sequence and shape [18]. While, they have been demonstrated to outperform commonly used position weight matrix-based binding models [19], they have received less attention by the deep learning community in genomics. Even though mono-nucleotide-based one-hot encoding approach captures higher-order sequence features to some extent by combining the sequence information in a complicated way through e.g. multiple convolutional layers [13], our results demonstrate that it is more effective to capture correlations between neighbouring nucleotides at the initial layer, rather than to defer this responsibility to subsequent convolutional layers.

Janggu also exposes variant effect prediction functionality, similar as Kipoi and Selene [10, 11], that allow to make use of the higher-order sequence encoding.

Conclusion

We present *Janggu*, a novel python library that facilitates deep learning in genomics.

- *Janggu* provides reusable dataset components that can be used in a unified and flexible way to set up deep learning applications in genomics in a variety of ways, which we have demonstrated in this article. These dataset objects are not tied to a specific machine learning library, but can be consumed with a range of popular frameworks, including keras or scikit-learn.

- Coverage tracks from BAM, BIGWIG or BED can be transformed and normalization. In addition, we have demonstrated that data augmentation might further improve the generalization of models based on coverage data.

- *Janggu* introduces higher-order sequence encoding based on e.g. di- or tri-nucleotides. We show that the new encoding generally improves the sequence based models.

- *Janggu* provides a number of evaluation-centered utilities that facilitate the interpretation of the models, including by inspection of the results in the built-in genome browser or by exporting the results to

BIGWIG files, by facilitating integrated gradient to highlights important input features or by screening for variants that potentially affect the features occurrence (e.g. TF binding). We expect improved accuracies for variant effect predictions through the use of higher-order sequence encodings.

Methods

Dataset and Evaluation for JunD prediction

We downloaded JunD peaks (ENCFF446WOD, conservative IDR thresholded peaks, narrowPeak format), and raw DNase-seq data (ENCFF546PJU, Stam. Lab, ENCODE; ENCFF059BEU Stam. Lab, ROADMAP, bam-format) for human embryonic stem cells (H1-hesc) from the encodeproject.org and the hg38 reference genome.

We defined all chromosomes as training chromosomes except for chr2 and chr3 which are used as validation and test chromosomes, respectively. The region of interest was defined as the union of all JunD peaks extended by 10kb with a binning of 200 bp. Each 200bp-bin is considered a positive labels if it overlaps with a JunD peak. Otherwise it is considered a negative example. For the DNA sequence, we further extended the context window by +/- 150 bp leading to a total window size of 500 bp. Similarly, for the DNase signal, we extracted the coverage in 50 bp resolution adding a flanking region of +/- 450 bp to each 200 bp window which leads to a total input window size of 1100bp. Dataset normalization and data augmentation was performed using *Janggu* dataset objects.

We implemented the architectures given in Table S.1 and S.2 for the individual models. The individual submodels were combined by removing the output layer, concatenating the top-most hidden layers and adding a new output layer.

Training was performed using a binary cross-entropy loss with AMSgrad [20] for at most 30 epochs using early stopping monitored on the validation set with a patience of 5 epochs. We trained each model 5 times with random initialization in order to assess reproducibility. performance were measured on the independent test chromosome using the area under the precision recall curve (auPRC).

Dataset and Evaluation of published models

Following the instructions of Zhou *et. al* [4], we downloaded the human genome hg19 and the same 919 features in narrowPeak format from ENCODE and ROADMAP and implemented the neural network architecture accordingly using *keras* and *janggu*.

All genomic regions used to train the original DeepSEA model (allTFs.pos.bed.tar.gz) were downloaded from http://deepsea.princeton.edu/. Following their procedure, all regions on chromosomes 8 and 9 were assigned to the test set, while the remaining regions were used for training and validation.

We downloaded the narrowPeak files from the URLs listed in Supplementary table 1 of Zhou *et. al* [4], adapting broken links where necessary.

We implemented DeepSEA as described in Zhou *et. al* [4] and DanQ as described in Quang *et. al* [17]. In addition, the models were adapted to scan both DNA strands rather than only the forward strand using the DnaConv2D layer, available in the *Janggu* library. For our investigation, we compared different context window sizes 500bp, 1000bp and 2000bp as well as mono-, di- and tri-nucleotide based sequence encoding.

The models were trained using AMSgrad [20] for at most 30 epochs using early stopping with a patience of 5 epochs.

We evaluated the performance using the auPRC on the independent test regions.

Dataset and Evaluation for CAGE-tag prediction

For the CAGE-tag prediction we focused on human HepG2 cells. We downloaded samples for CAGE (ENCFF177HHM, bam-format), DNase (ENCFF591XCX, bam-format) and H3K4me3 (ENCFF736LHE, bigwig-format) from the ENCODE project. Moreover, we used the hg38 reference genome and extracted the set of all protein coding gene promoter regions (200 bp upstream from the TSS) from GENCODE version V29 which constitute the ROI.

We loaded the DNA sequence using a +/- 350bp flanking window. For CAGE, DNase and H3K4me3, we summed the signal for each promoter using flanking windows of 400 bp, 200 bp and 200 bp to each dataset, respectively. The promoter signals for each feature were subsequently log-transformed using a pseudo-count of one and then Z score normalized.

The DNA and chromatin-based models are summarized in Tab. S.3 and S.4. Furthermore, the joint model is built by concatenating the top most hidden layers and adding a new output layer. We pursued a cross-validation strategies where we trained a model on genes of all chromosomes but one validation autosome, repeating the process for each autosome. Genes on chromosome 1 were left out entirely from the cross-validation runs and were used for the final evaluation. The models were trained using mean absolute error loss with AMSgrad [20] for at most 100 epochs using early stopping with a patience of 5 epochs.

For the evaluation of the model performance, we used the Pearson's correlation between the predicted and observed CAGE-signal on the test dataset.

Software availability

Janggu is freely available using the pypi echosystem and via github under a GPL-v3 license. A comprehensive documentation, including tutorials, can be found at https://janggu.readthedocs.io.

Competing interests

The authors declare that they have no competing interests.

Acknowledgements

The authors wish to thank Jonathan Ronen for valuable comments on the manuscript.

References

[1] Angermueller, C., Pärnamaa, T., Parts, L., Stegle, O.: Deep learning for computational biology. Molecular systems biology **12**(7), 878 (2016)

[2] Eraslan, G., Avsec, Ž., Gagneur, J., Theis, F.J.: Deep learning: new computational modelling techniques for genomics. Nature Reviews Genetics, 1 (2019)

[3] Alipanahi, B., Delong, A., Weirauch, M.T., Frey, B.J.: Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. Nature biotechnology **33**(8), 831 (2015)

[4] Zhou, J., Troyanskaya, O.G.: Predicting effects of noncoding variants with deep learning–based sequence model. Nature methods **12**(10), 931 (2015)

[5] Kelley, D.R., Snoek, J., Rinn, J.L.: Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks. Genome research (2016)

[6] Kelley, D.R., Reshef, Y.A., Bileschi, M., Belanger, D., McLean, C.Y., Snoek, J.: Sequential regulatory activity prediction across chromosomes with convolutional neural networks. Genome research **28**(5), 739–750 (2018)

[7] Singh, R., Lanchantin, J., Robins, G., Qi, Y.: Deepchrome: deep-learning for predicting gene expression from histone modifications. Bioinformatics **32**(17), 639–648 (2016)

[8] Angermueller, C., Lee, H.J., Reik, W., Stegle, O.: Deepcpg: accurate prediction of single-cell dna methylation states using deep learning. Genome biology **18**(1), 67 (2017)

[9] Budach, S., Marsico, A.: pysster: Classification of biological sequences by learning sequence and structure motifs with convolutional neural networks. Bioinformatics **1**, 3 (2018)

[10] Avsec, Z., Kreuzhuber, R., Israeli, J., Xu, N., Cheng, J., Shrikumar, A., Banerjee, A., Kim, D.S., Urban, L., Kundaje, A., et al.: Kipoi: accelerating the community exchange and reuse of predictive models for genomics. bioRxiv, 375345 (2018)

[11] Chen, K.M., Cofer, E.M., Zhou, J., Troyanskaya, O.G.: Selene: a pytorch-based deep learning library for sequence-level data. bioRxiv, 438291 (2018)

[12] Sundararajan, M., Taly, A., Yan, Q.: Axiomatic attribution for deep networks. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70, pp. 3319–3328 (2017). JMLR. org

[13] Greenside, P., Shimko, T., Fordyce, P., Kundaje, A.: Discovering epistatic feature interactions from neural network models of regulatory DNA sequences. Bioinformatics **34**(17), 629–637 (2018). doi:10.1093/bioinformatics/bty575.

http://oup.prod.sis.lan/bioinformatics/article-pdf/34/17/i629/25702257/bty575.pdf

[14] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research **15**(1), 1929–1958 (2014)

[15] Thurman, R.E., Rynes, E., Humbert, R., Vierstra, J., Maurano, M.T., Haugen, E., Sheffield, N.C., Stergachis, A.B., Wang, H., Vernot, B., *et al.*: The accessible chromatin landscape of the human genome. Nature **489**(7414), 75 (2012)

[16] Simard, P.Y., Steinkraus, D., Platt, J.C., *et al.*: Best practices for convolutional neural networks applied to visual document analysis. In: Icdar, vol. 3 (2003)

[17] Quang, D., Xie, X.: Danq: a hybrid convolutional and recurrent deep neural network for quantifying the function of dna sequences. Nucleic acids research **44**(11), 107–107 (2016)

[18] Zhou, T., Shen, N., Yang, L., Abe, N., Horton, J., Mann, R.S., Bussemaker, H.J., Gordân, R., Rohs, R.: Quantitative modeling of transcription factor binding specificities using dna shape. Proceedings of the National Academy of Sciences **112**(15), 4654–4659 (2015)

[19] Keilwagen, J., Grau, J.: Varying levels of complexity in transcription factor binding motifs. Nucleic acids research **43**(18), 119–119 (2015)

[20] Reddi, S.J., Kale, S., Kumar, S.: On the convergence of adam and beyond. In: International Conference on Learning Representations (2018). https://openreview.net/forum?id=ryQu7f-RZ

Figures

Tables

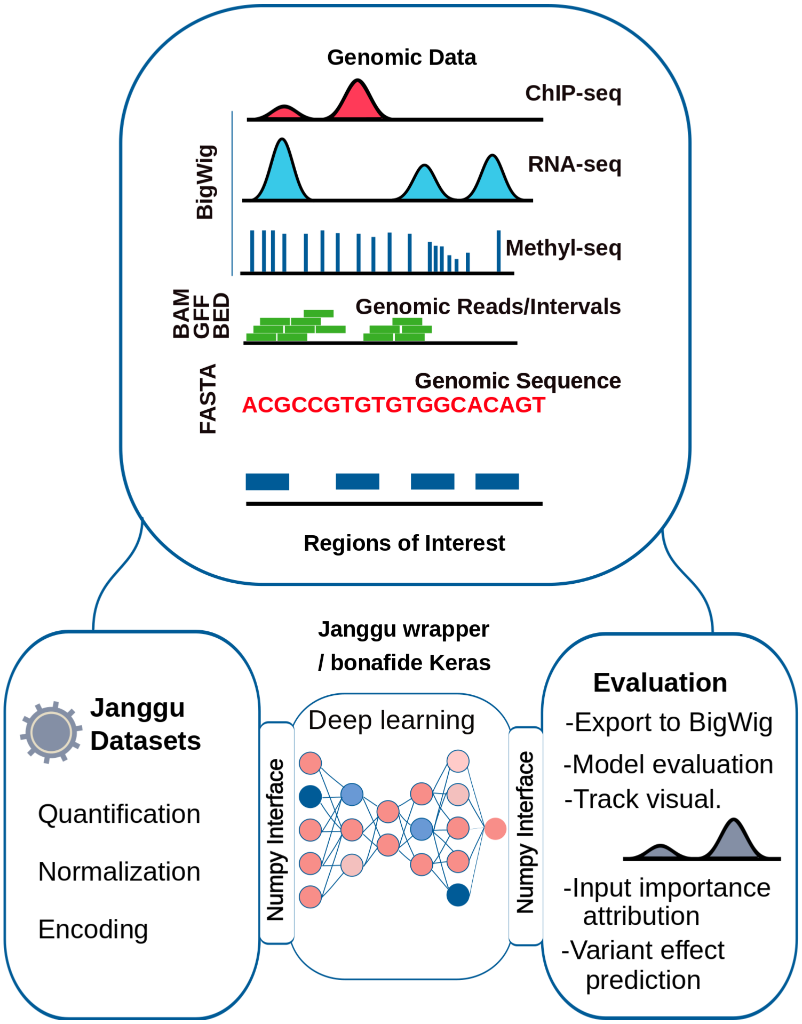

Figure 1: **Janggu schematic overview.** Janggu helps with data aquisition and evaluation of deep learning models in genomics. Data can be loaded from various standard genomics file formats, including FASTA, BED, BAM and BIGWIG. The output predictions can be converted back to coverage tracks and exported to BIGWIG files.

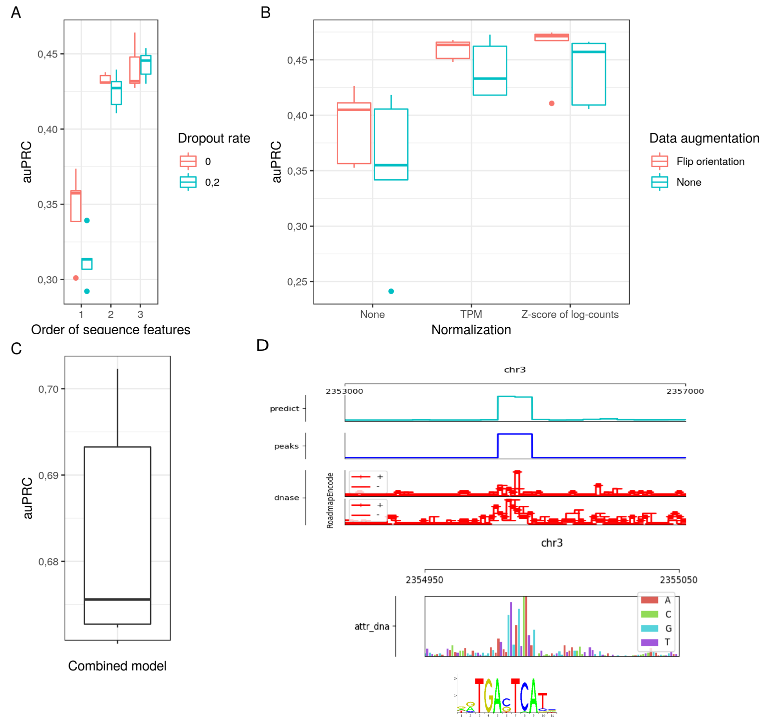

Figure 2: **Performance evaluation of JunD prediction.** A) Performance comparison of different one-hot encoding orders. Order 1, 2, and 3 correspond to mono-, di- and tri-nucleotide based one-hot encoding, respectively. B) Performance comparison of different normalization and data augmentation strategies applied to the read counts from the BAM files. Each model was trained from scratch for five times. We compared 1) No normalization (”None”), 2) TPM normalization, and 3) Z score of log(count + 1). Moreover, data augmentation consisted of 1) no augmentation (”None”) or 2) randomly flipping 5’ to 3’ orientations. Each model was trained from scratch for five times. C) Performance for JunD prediction for the combined model that takes DNA and DNase coverage into account. D) Example of a JunD binding site. The top most panel shows predicted, true JunD binding site as well as the input DNase coverage around the peak. Underneath integrated gradients further highlights the importance of a site reminiscent of the known JunD motif (Jaspar motif: MA091.1).

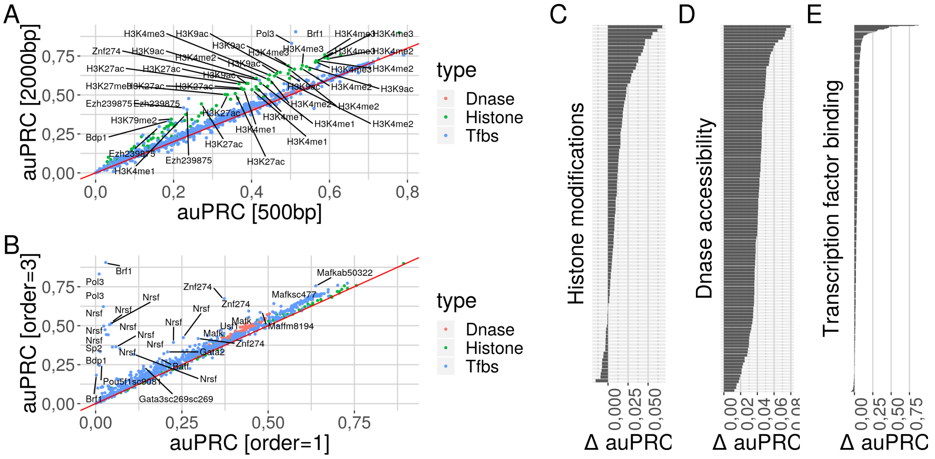

Figure 3: **Comparison of DanQ model variants.** A) auPRC comparison for the context window sizes 500bp and 2000bp for tri-nucleotide based sequence encoding. The mark color indicates the feature types: DNase hypersensitive sites, histone modifications and transcription factor binding assays. B) auPRC comparison for tri- and mono-nucleotide based sequence encoding for a context window of 2000bp. Color coding as above. C-E) Differences in auPRC between tri- and mono-nucleotides for histone modifications, DNase accessibility and transcription factor binding, respectively.

Table 1: Average Pearson's correlation across the cross-validation runs between observed and predicted normalized CAGE-counts. The models used either DNA or Chromatin (DNase and H3K4me3) or both simultaneously as input. Furthermore, different one-hot encoding orders were considered for the DNA sequence.

Model	DNA order	mean Pearson's corr.	Stand. Error
Chromatin only	-	0.777	2.97×10^{-5}
DNA only	1	0.533	4.38×10^{-3}
DNA only	2	0.559	8.40×10^{-3}
DNA only	3	0.585	6.47×10^{-3}
DNA & Chromatin	1	0.775	6.01×10^{-4}
DNA & Chromatin	2	0.783	5.13×10^{-4}
DNA & Chromatin	3	0.784	5.15×10^{-4}

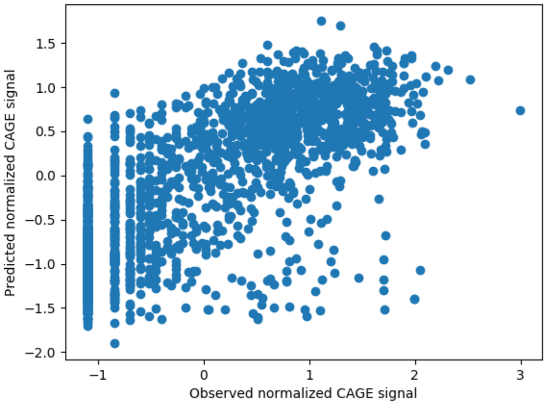

Figure 4: **Agreement between predicted and observed CAGE signal.** The example illustrates the agreement between predicted and observed CAGE signal on the test chromosome for the joint DNA-chromatin model. The DNA was represented as tri-nucleotide based one-hot encoding.

Supplementary Notes

July 15, 2019

Table S.1: DNA model for JunD prediction. DnaConv2D constitutes a wrapper that allows to scan both DNA strands with the same kernels.

Conv2D(10, (11, 1), ’relu’)
DnaConv2D()
MaxPool2D(30, 1)
BatchNormalization()
Conv2D(8, (3, 1), ’relu’)
GlobalMaxPooling()
BatchNormalization()
Dense(1, ’sigmoid’)

Table S.2: DNase model for JunD prediction.

Conv2D(10, (5, 2), ’relu’)
MaxPool2D(2, 1)
BatchNormalization()
Conv2D(5, (3, 1), ’relu’)
GlobalMaxPooling()
BatchNormalization()
Dense(1, ’sigmoid’)

Table S.3: DNA model for CAGE-tag prediction. λ was set to 0.0 and 0.2 for order one or higher order sequence features (two and three).

Dropout(λ)
Conv2D(10, (15, 1), 'relu')
MaxPool2D(5, 1)
BatchNormalization()
Conv2D(8, (5, 1), 'relu')
GlobalMaxPooling()
BatchNormalization()
Dense(1, 'linear')

Table S.4: Chromatin model for CAGE-tag prediction.

Concatenate()([dnase_signal, h3k4me3_signal])
BatchNormalization()
Dense(1, 'linear')

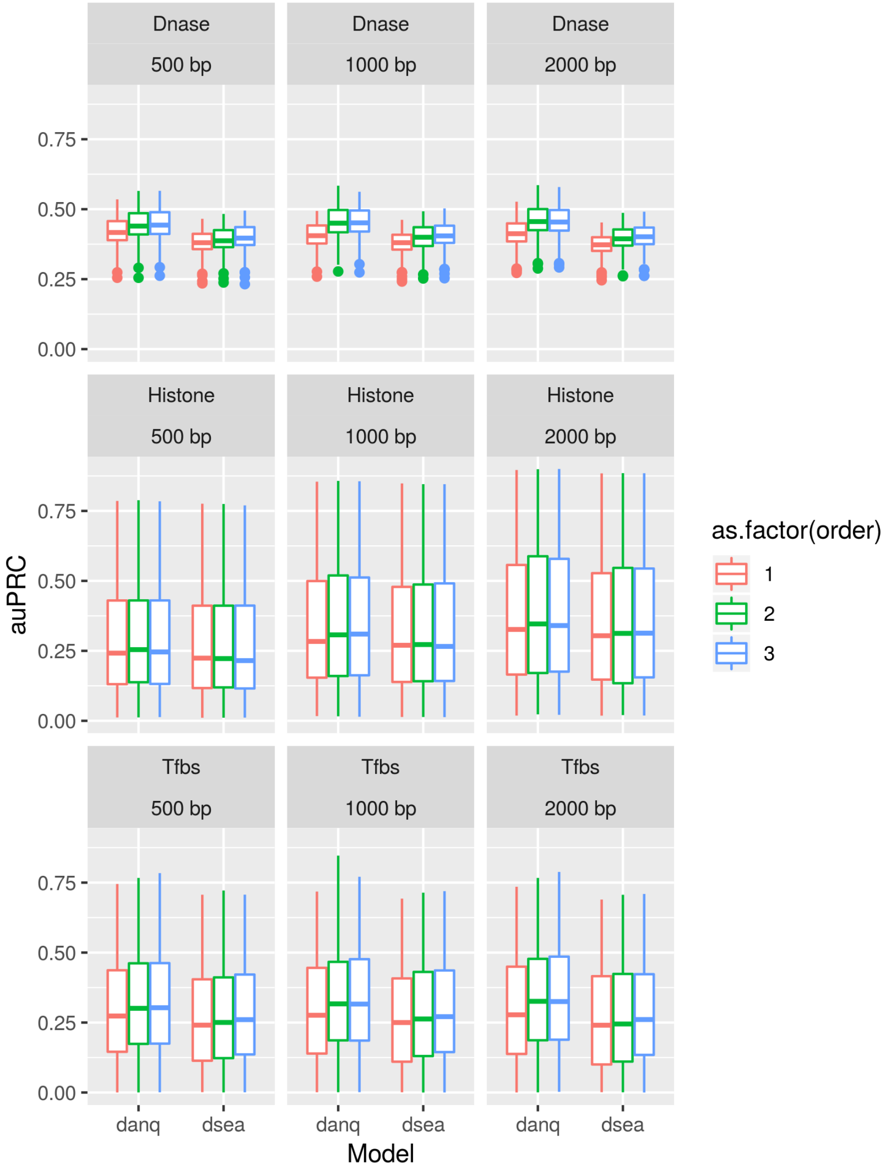

Figure S.1: **DeepSEA and DanQ comparison.** Performance comparison of DeepSEA and DanQ on the same benchmark data measured by the area under the precision-recall curve. The comparison dissects performances for different genomic features (Dnase, histone modifications and TF binding sites), different context window sizes (500bp, 1000bp and 2000bp) and different sequence encoding orders (order one, two and three).