layerUMAP: A tool for visualizing and understanding deep learning models in biological sequence classification using UMAP

Runyu Jing, Li Xue, Menglong Li, Lezheng Yu, Jiesi Luo

xinyan_scu@126.com (L.Y.)
ljs@swmu.edu.cn (J.L.)

Highlights
We developed layerUMAP for the visualization and interpretation of deep learning models

LayerUMAP can investigate high-level representations learned by the models

LayerUMAP can dissect the deep learning model layer by layer interactively

Jing et al., iScience 25, 105530
December 22, 2022 © 2022
The Author(s).
https://doi.org/10.1016/j.isci.2022.105530

OPEN ACCESS
layerUMAP: A tool for visualizing and understanding deep learning models in biological sequence classification using UMAP

Runyu Jing, Li Xue, Menglong Li, Lezheng Yu, and Jiesi Luo

SUMMARY
Despite the impressive success of deep learning techniques in various types of classification and prediction tasks, interpreting these models and explaining their predictions are still major challenges. In this article, we present an easy-to-use command line tool capable of visualizing and analyzing alternative representations of biological observations learned by deep learning models. This new tool, namely, layerUMAP, integrates autoBioSeqpy software and the UMAP library to address learned high-level representations. An important advantage of the tool is that it provides an interactive option that enables users to visualize the outputs of hidden layers along the depth of the model. We use two different classes of examples to illustrate the potential power of layerUMAP, and the results demonstrate that layerUMAP can provide insightful visual feedback about models and further guide us to develop better models.

INTRODUCTION
Deep learning (DL) models have recently achieved breakthrough performances on a variety of difficult tasks, including life science tasks. Unfortunately, for nonexpert users, their accessibility is often fraught with technical challenges. As most DL methods are available as source code, running them requires setting up a sophisticated software and hardware environment. The increasing use of next-generation sequencing analysis workflows in bioinformatics and computational biology and the willingness to propagate well-trained DL models are pushing computer scientists to design more user-friendly solutions. An increasing number of libraries or tools addressing this problem with different solutions have become available, including Kipoi, DragoNN, Selene, pyßer, SECLAF, Janggu, BioSeq-BLM, and autoBioSeqpy. Kipoi offers more than 2,000 individually trained models from 22 distinct studies, enabling users to perform a variety of canonical genomic predictive tasks using trained DL models. Through its API, it is possible to share or reuse trained models on a local machine without any previous programming skills. DragoNN provides an early release DL toolkit that is equipped with both inference and training functionalities for regulatory genomics. Selene is a PyTorch-based library for developing, training and applying DL model architectures to biological sequence classification and making predictions by writing a few simple command-line commands. pyßer is particularly suited for building and training CNNs for RNA sequence classification by learning structural motifs corresponding to sequence motifs. The trained residue-sequence models of SECLAF have been made available in a preconfigured webserver. Janggu combines dedicated dataset objects provided by Bioseq and Cover with popular DL libraries, such as Keras and PyTorch, to simplify DL applications in genomics with reusable components. BioSeq-BLM utilizes browser user interfaces to train different models in a noncoding fashion, together with a set of 155 biological language representations that support the easy training and use of their models. autoBioSeqpy is a contribution of our group to the field that increases the accessibility of DL for biological sequences. The main features and characteristics of autoBioSeqpy are summarized as follows: (1) Model development is separated into independent parts, enabling users to design and tune models flexibly according to their needs; (2) all parts, including dataset reading, sequence encoding, model training and evaluation, can be executed with command-line tools, enabling users to easily modify the parameters of the entire workflow; and (3) it provides documentation in a browser-based notebook interface, enabling users to further analyze the models.

The previously described tools have started to boost the use of DL solutions for biological sequence prediction tasks and facilitate the dissemination of DL models in a standardized manner. Despite this
encouraging progress, there is still little insight into the inner operation and behavior of DL models or how they perform well. Typically, DL models are treated as “black box” function approximators that map a given input to a classification output. Because of this “black box” characteristic, the development of high-quality DL models usually relies on a time-consuming trial-and-error procedure. Techniques for understanding the mechanism and the performance of DL models have therefore become a key ingredient of a robust validation procedure. Recently, UMAP and t-SNE have been frequently used together with deep learning methods to understand what a model might have learned. However, a user-friendly tool for integrating these techniques to visualize the hidden activities of deep learning models in a noncoding fashion and with a unified format is still missing. We present layerUMAP, a command-line tool included in the autoBioSeqpy software, written in Python, which makes use of UMAP to understand, diagnose and refine DL models. layerUMAP provides access to the hidden layers of trained models developed by autoBioSeqpy. Hence, users can easily perform two valuable visual feedback tasks: investigating high-level representations learned by the models and dissection each model’s performance layer by layer. This is especially helpful for users to compare and select models based on their data, making the dissemination of DL models more effective. We demonstrate layerUMAP on two use cases: (1) Classifying Gram-negative bacterial secreted proteins from primary sequences and (2) identifying cell types from single-cell gene expression data. In these examples, different types of DL models are developed, including convolutional, recurrent, convolutional-recurrent and deep neural networks, which cover the most commonly used network architectures. This study highlights the use of layerUMAP for the improved visualization and interpretation of deep learning models.

RESULTS
In this section, the metrics that were used to evaluate the predictions from the models are provided. The following subsections present the two case studies conducted in this work.

Case study 1: Classifying Gram-negative bacterial secreted proteins from primary sequences
We performed a grid search to exhaustively test hyperparameter combinations of the number of convolution layers (1, 2, 3, and 4), convolution filter length (3, 5, 7, 9, 11, and 13), number of convolution filters (50, 100, 150, 200, 250, and 300), number of BiLSTM layers (1, 2, 3, and 4), number of BiGRU layers (1, 2, 3, and 4), number of nodes in the BiLSTM layer (16, 32, 64, and 128), number of BiGRU layers (16, 32, 64, and 128). In the process of tuning the hyperparameters, we varied each hyperparameter individually while keeping the remaining hyperparameters constant. We measured the performance change with respect to the change in each hyperparameter by repeating the training procedure 10 times and obtained 10 trained models. During testing, the performance of each hyperparameter was evaluated using all 10 trained models, and the accuracy over these 10 models was used as the final score. The optimization results for various hyperparameters are shown in Figure 1. Because the performance of the convolution operation depends critically on the length and number of filters, we evaluated a range of filter lengths from 3 to 13 in steps of 2 and a range of filter numbers from 50 to 300 in steps of 50. Figure 1A shows that the accuracy of the CNN model varied substantially as the relevant hyperparameters were varied. The combination of hyperparameters that yielded the highest classification accuracy was filter length = 5 and number of filters = 300. We also found that the performance of the CNN model was significantly affected by the pooling strategy. Global max pooling yielded notably better results than max pooling (Figure 1B). This may have been because global max pooling extracts information from the convolved sequences more effectively than max pooling. For two special types of RNNs, we observed that LSTM and GRU performed poorly compared to BiLSTM and BiGRU, respectively, which is consistent with our previous findings (Figure 1C). The bidirectional approach can learn the context features from input sequences in both directions (forward and backwards) and provide additional information about the residues in the sequence before and after the current position. Furthermore, we varied the input sizes of the BiLSTM and BiGRU layers from 16 to 32, 64, and 128 to examine their influence on the classification accuracy. With an input size of 64 (a hidden size of 128 for the bidirectional layer), both RNN models showed the best performance (Figure 1D). We varied the number of different types of network layers to further investigate the impact of network depth on model performance. When we increased the number of layers from 1 to 4, the performance improved with increasing depth, and the highest classification accuracy was reached with three recurrent layers for BiLSTM and BiGRU (Figure 1D). In contrast, for CNN, although increasing depth was initially beneficial, it had the opposite effect after two convolutional layers. These
observations are summarized in a schematic illustration of the proposed deep learning architectures in Figure 1F.

We implemented and trained the above five DL models in autoBioSegpy using the command-line command 1, as shown in the Figure 2. Here, we used the CNN model as an example. The classwise prediction
CNN and CNN-BiGRU achieved a high accuracy of 97.9%. The other models also showed relatively high accuracies, of 97.7% for CNN-BiLSTM, 97.3% for BiLSTM and 95.8% for BiGRU.

Case study 2: Identifying cell types from single-cell gene expression data

Here, we developed a novel deep learning model called RFCN for the type classification of single cells using residual fully connected networks. The overall model architecture is illustrated in Figure 4A. The input to the model is a matrix of gene expression values measured in a scRNA-seq experiment, with each column representing a gene and each row representing a cell. We applied the RFCN to three scRNA-seq datasets from pancreatic tissue, PBMCs and lung cancer cells and showed that the RFCN was able to classify individual cells with high accuracy (Figure 4B). Specifically, RFCN achieved overall accuracies of 98.09, 99.78 and 100% in cell type classification on the Zheng sorted, Pancreas and CellBench datasets, respectively.

Instead of only considering the accuracy, we also evaluated the F1-score for each cell population. The median of these F1-scores was used as the final measure for the performance on the dataset. On all three datasets, RFCN also performed well, with a median F1-score >0.98. We further visualized representations of cells learned by RFCN in the two-dimensional UMAP space using layerUMAP.

Command-line command 1:
```
python running.py --dataType protein --dataEncodingType onehot --dataTrainFilePaths /data/T15S_train.txt /data/T25S_train.txt /data/T35S_train.txt /data/T45S_train.txt /data/T55S_train.txt /data/T75S_train.txt --dataTestFilePaths /data/T15S_test.txt /data/T25S_test.txt /data/T35S_test.txt /data/T45S_test.txt /data/T55S_test.txt /data/T75S_test.txt --dataTestLabel 0 1 2 3 4 5 --modelLeadFile /model/CNN.py --version 1 --outSaveFolderPath tmpOut --savePrediction 1 --saveFig 1 --batch_size 40 --epochs 50 --spclLen 2000 --shuffleDataTrain 1 --modelSaveName tmpMod.json --weightSaveName tmpWeight.bin --noGPU0 --paraSaveName parameters.txt --labelTrainMat 1 --loss categorical_crossentropy
```

Command-line command 2:
```
python tool/layerUMAP.py --paraFile tmpOut/parameters.txt --outFigFolder tmpOut --metric chebychev --theme fire --n_neighbors 28 --min_dist 0.9
```

Command-line command 3:
```
python tool/layerUMAP.py --paraFile tmpOut/parameters.txt --outFigFolder tmpOut --metric chebychev --theme fire --n_neighbors 28 --min_dist 0.9 --interactive 1
```

Command-line command 4:
```
python tool/layerUMAP.py --paraFile tmpOut/parameters.txt --outFigFolder tmpOut --metric chebychev --interactive 1 --grid_n_neighbors 2 30 --grid_min_dist 0.1 1 0.1
```

Command-line command 5:
```
python tool/layerUMAP.py --paraFile tmpOut/parameters.txt --outFigFolder tmpOut --metric cosine --n_neighbors 28 --min_dist 0.8
```

Figure 2. Example commands illustrate how to use autoBioSeqpy and layerUMAP for deep learning model analysis

The quality of each model is shown in the confusion matrix in Figure 3A. CNN and CNN-BiGRU achieved a high accuracy of 97.9%. The other models also showed relatively high accuracies, of 97.7% for CNN-BiLSTM, 97.3% for BiLSTM and 95.8% for BiGRU.

Case study 2: Identifying cell types from single-cell gene expression data

Here, we developed a novel deep learning model called RFCN for the type classification of single cells using residual fully connected networks. The overall model architecture is illustrated in Figure 4A. The input to the model is a matrix of gene expression values measured in a scRNA-seq experiment, with each column representing a gene and each row representing a cell. We applied the RFCN to three scRNA-seq datasets from pancreatic tissue, PBMCs and lung cancer cells and showed that the RFCN was able to classify individual cells with high accuracy (Figure 4B). Specifically, RFCN achieved overall accuracies of 98.09, 99.78 and 100% in cell type classification on the Zheng sorted, Pancreas and CellBench datasets, respectively. Instead of only considering the accuracy, we also evaluated the F1-score for each cell population. The median of these F1-scores was used as the final measure for the performance on the dataset. On all three datasets, RFCN also performed well, with a median F1-score >0.98. We further visualized representations of cells learned by RFCN in the two-dimensional UMAP space using layerUMAP.
This section discusses the performance of the best model demonstrated in the ‘results’ section by using plots generated by layerUMAP. A practical explanation of the neural layers from the built model together with the related interpretation are presented according to the two case studies.

**Case study 1: Classifying Gram-negative bacterial secreted proteins from primary sequences**

To further validate these results, we visualized the last hidden layer representations of the DL models using the layerUMAP tool with the command-line command 2 (Figure 2). If the ‘--interactive’ option is not used, layerUMAP will output the projection of the last hidden layer of the trained model by default, and ‘tmpOut/parameters.txt’ is the parameter file, which can be generated by command-line command 1. The runtime of the above command was approximately 30 s, excluding the interaction time. Each point in Figure 3B represents a Gram-negative bacterial secreted protein projected from the 6-dimensional output of the model’s last hidden layer into two dimensions. Colored point clouds represent the different secretory types, showing how the model clusters the proteins. We see that points of the same category are clustered together and clearly separated from points of other categories. When we investigated the number of misclassified proteins across all classes for the best CNN model, we observed that a total of 14 proteins were not correctly recognized by the network: 1 originated from T1SS, three from T2SS, four from T4SS, two from T5SS and four from T7SS. All proteins originating from T3SS were classified correctly. Further investigating each cluster, we found that 4 T4SS, 2 T2SS and 1 T7SS protein were classified as T3SS proteins (Cluster 2); 2 T7SS and 1 T5SS protein were classified as T2SS proteins (Cluster 1); 1 T2SS and 1 T5SS protein were classified as T4SS proteins (Cluster 3); and 1 T1SS and 1 T7SS protein were classified as T5SS proteins (Cluster 4), which is consistent with the classification results of the CNN model. We next evaluated the models in terms of test accuracy, architectural complexity and computational complexity. Considering the number of parameters, the CNN-BiLSTM model was found to have the most complex architecture, with a total of 952,868 trainable parameters (Figure 3C). Although achieving comparable accuracies, the BiGRU model had fewer parameters than CNN-BiLSTM. CNN was the fastest model with respect to runtimes and finished an epoch in under 20 s, which was much faster than the other models.
less than the times required by other models (Figure 3D). With regard to test accuracy, CNN also outperformed the other competing models, yielding the highest classification accuracy (91.3%) (Figure 3E). The next-best-performing model was CNN-BiGRU (89.5%), followed by BiLSTM (86.6%). In conclusion, our results demonstrate that the proposed CNN architecture is the most appropriate model for distinguishing between different types of Gram-negative bacterial secreted proteins.

To further showcase the ability of layerUMAP to analyze and diagnose DL models, we performed a layer-by-layer dissection of the CNN architecture using the command-line command 3 (Figure 2). layerUMAP gives the user the flexibility to run UMAP routines using a range of options. For example, by using the layer names or indices, the user can explore the layers of interest in more depth. It is clear that the data containing more than three dimensions cannot be directly plotted into a 2D figure, which limits the visualization in a few layers, such as the reshape layers and convolution layers. To address this issue, we provided several dimensionality reduction options for reducing multi-dimensional data into lower-dimensional data. To facilitate understanding, we considered the dimensionality reduction of the convolution layer as an example to illustrate how layerUMAP performs these operations. As shown in Figure 5, by calculating the possible values (maximum, minimum or average) for each row or column, the feature matrix of the sample was compressed in two directions, namely, ‘Filtered sequence length’ and ‘Channel’. Samples generated in the ‘Filtered sequence length’ direction were mixed together due to the loss of much channel information, while the other direction retained more channel information, resulting in a better separation. In addition, each
channel could be extracted independently for UMAP plotting, which facilitated the comparison of filters. Finally, we used layerUMAP to observe how the internal representations of samples evolved in different network layers (Figure 6). For the first two convolutional layers, we applied the dimensionality reduction procedure to the data by computing the maximum values in the ‘Channel’ direction.

Case study 2: Identifying cell types from single-cell gene expression data
To investigate the effects of two UMAP parameters (n_neighbors and min_dist) on the resulting embedding, we first performed a grid search in the parameter space using the command-line command 4 (Figure 2). The runtime of this command without interaction was approximately 24 min because we conducted a grid search with ‘n_neighbors’ and ‘min_dist’ in 126 iterations; when the grid search was turned off, the runtime was reduced to approximately 50 s. This 50-s process was roughly divided into two parts, one concerning the preparation of the temporary model and the generation of the output of the middle layer and the other calling the UMAP library and plotting. According to our tests, the first part took approximately 39 s, and the second part took approximately 11 s.

Using the Zheng sorted dataset as an example, we observed that the parameter ‘min_dist’ had a significant impact on the separation of cell populations. With an increasing value of min_dist, UMAP ultimately separated cell populations into distinct clusters that corresponded to their cell types (Figure S1). Furthermore, we compared the effects of ten different metric parameters: chebyshev, braycurtis, canberra, correlation, cosine, euclidean, mahalanobis, manhattan, minkowski and wminkowski. Only wminkowski failed to separate cell populations (Figure S2). The other metrics revealed relatively similar clusters, with a few differences. After optimizing the UMAP parameters, we used the command-line command 5 to visualize the classification results of the RFCN model (Figure 2). RFCN learned to embed cells of the same type close to each other while embedding cells of different types far from each other, agreeing well with the cell type annotations (Figure 4C).

Limitations of the study
layerUMAP was found to provide a convenient method for the hidden data visualization of a deep learning model. However, the architecture of a deep learning model can be very complex, e.g., with distillation and multi-task learning structures. Currently, layerUMAP cannot capture the hidden outputs from the layers in a branch (if existing) of the built model, and users can only capture the outputs manually using the related Jupyter Notebook.

Conclusions
In recent years, there has been an explosion of deep learning applications in many aspects of biological research. A variety of biological data can be modeled using deep learning techniques, including DNA, RNA,21,22 and protein data.2,23 Despite the great success of deep learning in solving various pattern
recognition tasks in recent years, the problem of understanding the mechanisms of DL models remains largely unresolved and, thus, has become an issue in the development of DL. Without a deeper understanding of the method, the development of high-quality DL models relies on a substantial number of trial-and-error procedures, and the use of trained models could also potentially become a source of unreliable results. Because DL models consist of numerous layers, functional components and operations, interpreting them is still a technical challenge. To address this challenge, we have developed an interactive, visual analytics tool called layerUMAP, which aims to help DL users better design and diagnose DL models. We showed how layerUMAP can be used to understand the mechanism and the performance of a model and to inspect the model’s inner status at the layer level. We successfully applied this tool in our previous studies.\textsuperscript{24,25} To further demonstrate the capabilities of layerUMAP, we presented two case studies in this article. We expect layerUMAP to provide insight into the mechanisms of DL models and to become a standard tool for the visual analysis of DL models.

**STAR METHODS**

Detailed methods are provided in the online version of this paper and include the following:

- **KEY RESOURCES TABLE**
- **RESOURCE AVAILABILITY**
  - Lead contact
  - Materials availability
  - Data and code availability
- **METHOD DETAILS**
  - Overview of layerUMAP
  - Dataset for case study 1
  - Deep learning architectures for case study 1
  - Datasets for case study 2
  - Deep learning architectures for case study 2
- **QUANTIFICATION AND STATISTICAL ANALYSIS**

**SUPPLEMENTAL INFORMATION**

Supplemental information can be found online at https://doi.org/10.1016/j.isci.2022.105530.
ACKNOWLEDGMENTS

This work has been supported by the National Natural Science Foundation of China (No. 21803045 and No. 22173065), Joint project of Luzhou Municipal People’s Government and Southwest Medical University (2020LZXNYDJ39), and Southwest Medical University Research Program (2020ZRQNB057). The numerical calculations in this paper have been done on Hefei advanced computing center.

AUTHOR CONTRIBUTIONS

J.L. and L.Y conceived the study and wrote the manuscript. R.J. contributed to the design, implementation and testing of the tool. L.X. and J.L. performed the data analysis. All authors read and approved the final manuscript.

DECLARATION OF INTERESTS

The authors have declared no competing interests.

INCLUSION AND DIVERSITY

We support inclusive, diverse, and equitable conduct of research.

REFERENCES

1. Alipanahi, B., Delong, A., Weirauch, M.T., and Frey, B.J. (2015). Predicting the sequence specificities of DNA and RNA-binding proteins by deep learning. Nat. Biotechnol. 33, 831–838.
2. Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Zidek, A., Potapenko, A., et al. (2021). Highly accurate protein structure prediction with AlphaFold. Nature 596, 583–589.
3. Gómez-de-Mariscal, E., García-López-de-Haro, C., Ouyang, W., Donats, L., Lundberg, E., Unser, M., Muñoz-Barrutia, A., and Sage, D. (2021). DeepImageJ: a user-friendly environment to run deep learning models in ImageJ. Nat. Methods 18, 1192–1195.
4. Avsec, Z., Kreuzhuber, R., Israeli, J., Xu, N., Cheng, J., Shrikumar, A., Banerjee, A., Kim, D.S., Beier, T., Urban, L., et al. (2019). The Kipoi repository accelerates community exchange and reuse of predictive models for genomics. Nat. Biotechnol. 37, 592–600.
5. Movva, R., Greenside, P., Mannov, G.K., Nair, S., Shrikumar, A., and Kundaje, A. (2019). Deciphering regulatory DNA sequences and noncoding genetic variants using neural network models of massively parallel reporter assays. PLoS One 14, e0218073.
6. Chen, K.M., Cofer, E.M., Zhou, J., and Troyanskaya, O.G. (2019). Selene: a PyTorch-based deep learning library for sequence data. Nat. Methods 16, 315–318.
7. Budach, S., and Marsico, A. (2018). Pystser: classification of biological sequences by learning sequence and structure motifs with convolutional neural networks. Bioinformatics 34, 3035–3037.
8. Szalai, B., and Grohmsz, V. (2018). SECLAF: a webserver and deep neural network design tool for hierarchical biological sequence classification. Bioinformatics 34, 2487–2489.
9. Kopp, W., Monti, R., Tamburini, A., Ohler, U., and Akalin, A. (2020). Deep learning for genomics using Janggu. Nat. Commun. 11, 3488.
10. Li, H.-L., Pang, Y.-H., and Liu, B. (2021). BioSeq-BLM: a platform for analyzing DNA, RNA and protein sequences based on biological language models. Nucleic Acids Res. 49, e129.
11. Jing, R., Li, Y., Xue, L., Liu, F., Li, M., and Luo, J. (2020). autoBioSeqpy: a deep learning tool for the classification of biological sequences. J. Chem. Inf. Model. 60, 3755–3764.
12. Cock, P.J.A., Antao, T., Chang, J.T., Chapman, B.A., Cox, C.J., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B., et al. (2009). Biopython: freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics 25, 1422–1423.
13. Chollet, F. (2015). Keras: the Python deep learning API [WWW Document]. https://github.com/fchollet/keras.
14. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: an imperative style, high-performance deep learning library. Preprint at arXiv. https://doi.org/10.48550/arXiv.1912.01703.
15. Montavon, G., Samek, W., and Müller, K.R. (2018). Methods for interpreting and understanding deep neural networks. Digit. Signal Process. 73, 1–15.
16. McInnes, L., Healy, J., and Melville, J. (2018). Umap: uniform manifold approximation and projection for dimension reduction. Preprint at arXiv. https://doi.org/10.48550/arXiv.1802.03426.
17. Van der Maaten, L., and Hinton, G. (2008). Visualizing data using t-SNE. J Mach. Learn. Res. 9, 2579–2605.
18. Le, N.Q.K., Ho, Q.-T., Nguyen, T.-T.-D., and Ou, Y.-Y. (2021). A transformer architecture based on BERT and 2D convolutional neural network to identify DNA enhancers from sequence information. Brief. Bioinform. 22, bbab005.
19. Zhang, J.X., Yordanov, B., Gaunt, A., Wang, M.X., Dai, P., Chen, Y.-J., Zhang, K., Fang, J.Z., Dalichau, N., Li, J., et al. (2021). A deep learning model for predicting next-generation sequencing depth from DNA sequence. Nat. Commun. 12, 4387.
20. Le, N.Q.K., and Ho, Q.-T. (2022). Deep transformers and convolutional neural network in identifying DNA N6-methyladenine sites in cross-species genomes. Methods 204, 199–206.
21. Amin, N., McGrath, A., and Chen, Y.-P.P. (2019). Evaluation of deep learning in non-coding RNA classification. Nat. Mach. Intell. 1, 246–256.
22. Huang, Y., Li, Y., Liu, Y., Zhu, X., Jing, R., and Li, M. (2020). Multi-models in predicting RNA solvent accessibility exhibit the contribution from non-sequential attributes and providing a globally stable modeling
strategy. Chemometr. Intell. Lab. Syst. 205, 104100.

23. Tng, S.S., Le, N.Q.K., Yeh, H.-Y., and Chua, M.C.H. (2021). Improved prediction model of protein lysine Crotionylation sites using bidirectional recurrent neural networks. J. Proteome Res. 21, 265–273.

24. Yu, L., Xue, L., Liu, F., Li, Y., Jing, R., and Luo, J.L. (2022). The applications of deep learning algorithms on in silico druggable proteins identification. J. Adv. Res. 47, 219–231.

25. Yu, L., Liu, F., Li, Y., Luo, J., and Jing, R. (2021). DeepT3:4 a hybrid deep neural network model for the distinction between bacterial type III and IV secreted effectors. Front. Microbiol. 12, 605782.

26. Yu, L., Luo, J., Guo, Y., Li, Y., Pu, X., and Li, M. (2013). In silico identification of Gram-negative bacterial secreted proteins from primary sequence. Comput. Biol. Med. 43, 1177–1181.

27. Abbedalal, T., Michielsen, L., Cats, D., Hoogduin, D., Mei, H., Reinders, M.J.T., and Mahfouz, A. (2019). A comparison of automatic cell identification methods for single-cell RNA sequencing data. Genome Biol. 20, 194.

28. Hicks, S.W., Charon, G., Hang, H.C., and Galán, J.E. (2011). Subcellular targeting of Salmonella virulence proteins by host-mediated S-palmitoylation. Cell Host Microbe 10, 9–20.

29. Desvaux, M., Hebraud, M., Talon, R., and Henderson, I.R. (2009). Secretion and subcellular localizations of bacterial proteins: a semantic awareness issue. Trends Microbiol. 17, 139–145.

30. Lasica, A.M., Gsiazek, M., Madej, M., and Potempa, J. (2017). The type IX secretion system (T9SS): highlights and recent insights into its structure and function. Front. Cell. Infect. Microbiol. 7, 215.

31. Barouch, A., and Apteveier, R. (1999). The SWISS-PROT protein sequence data bank and its supplement TrEMBL in 1999. Nucleic Acids Res. 27, 49–54.

32. The UniProt Consortium (2007). The universal protein resource (UniProt). Nucleic Acids Res. 35, D190–D195.

33. Pruitt, K.D., Tatusova, T., Klimke, W., and Maglott, D.R. (2009). NCBI Reference Sequences: current status, policy and new initiatives. Nucleic Acids Res. 37, D32–D36.

34. Altschul, S.F., Madden, T.L., Schaffer, A.A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D.J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res. 25, 3389–3402.

35. Trieu, T., Martinez-Fundichely, A., and Khurana, E. (2020). DeepMILO: a deep learning approach to predict the impact of non-coding sequence variants on 3D chromatin structure. Genome Biol. 21, 79.

36. Alley, E.C., Khimulya, G., Biswas, S., AlQuareshi, M., and Church, G.M. (2019). Unified rational protein engineering with sequence-based deep representation learning. Nat. Methods 16, 1315–1322.

37. Wu, K.E., Fazal, F.M., Parker, K.R., Zou, J., and Chang, H.Y. (2020). RNA-GPS predicts SARS-CoV-2 RNA residency to host mitochondria and nucleolus. Cell Syst. 11, 102–108.e3.

38. Ding, J., Adiconis, X., Simmons, S.K., Kowalczyk, M.S., Hession, C.C., Marjanovic, N.D., Hughes, T.K., Wadsworth, M.H., Burks, T., Nguyen, L.T., et al. (2020). Systematic comparison of single-cell and single-nucleus RNA-sequencing methods. Nat. Biotechnol. 38, 737–746.

39. Miao, Z., Moreno, P., Huang, N., Papathodorou, I., Braze, A., and Teichmann, S.A. (2020). Putative cell type discovery from single-cell gene expression data. Nat. Methods 17, 621–628.

40. Cao, J., Spielmann, M., Qiu, X., Huang, X., Ibrahim, D.M., Hill, A.J., Zhang, F., Mundlos, S., Christiansen, L., Steemers, F.J., et al. (2019). The single-cell transcriptional landscape of mammalian organogenesis. Nature 566, 496–502.

41. Regev, A., Teichmann, S.A., Lander, E.S., Amit, I., Benoist, C., Birney, E., Bodenmiller, B., Campbell, P., Carninci, P., Clatworthy, M., et al. (2017). Science forum: the human cell atlas. Elife 6, e27041.

42. Hu, J., Li, X., Hu, G., Lyu, Y., Susztak, K., and Li, M. (2020). Iterative transfer learning with neural network for clustering and cell type classification in single-cell RNA-seq analysis. Nat. Mach. Intell. 2, 607–618.

43. Haghverdi, L., Lun, A.T.L., Morgan, M.D., and Marioni, J.C. (2018). Batch effects in single-cell RNA-sequencing data are corrected by matching mutual nearest neighbors. Nat. Biotechnol. 36, 421–427.

44. He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778.

45. Li, J., Miao, B., Wang, S., Dong, W., Xu, H., Si, C., Wang, W., Duan, S., Lou, J., Bao, Z., et al. (2022). Hiplot: a comprehensive and easy-to-use web service for boosting publication-ready biomedical data visualization. Brief. Bioinform. 23, bbac261. https://doi.org/10.1093/bib/bbac261.
STAR METHODS

KEY RESOURCES TABLE

| REAGENT or RESOURCE | SOURCE | IDENTIFIER |
|---------------------|--------|------------|
| Software and algorithms | | |
| autoBioSeqpy | Jing et al.\textsuperscript{11} | https://doi.org/10.1021/acs.jcim.0c00409 https://github.com/jingry/autoBioSeqpy/tree/2.0 |
| layerUMAP | This paper | https://github.com/jingry/autoBioSeqpy/blob/2.0/tool/layerUMAP.py https://github.com/jingry/autoBioSeqpy/blob/2.0/examples/layerUMAP.zip |
| Deposited data | | |
| Gram-negative bacterial secreted proteins dataset | Yu et al.\textsuperscript{26} | https://doi.org/10.1016/j.compbiomed.2013.06.001 http://cic.scu.edu.cn/bioinformatics/secretPv2_1/index.htm |
| scRNA-seq datasets | Abdelaal et al.\textsuperscript{27} | https://doi.org/10.1186/s13059-019-1795-z https://zenodo.org/record/3357167 |

RESOURCE AVAILABILITY

Lead contact
Further information and requests for resources and reagents should be directed to and will be fulfilled by the lead contact, Jiesi Luo (ljs@swmu.edu.cn).

Materials availability
This study did not generate new unique reagents.

Data and code availability
- This paper analyses existing, publicly available data. Accession numbers are listed in the key resources table, and the weblinks are http://cic.scu.edu.cn/bioinformatics/secretPv2_1/index.htm and https://zenodo.org/record/3357167.
- The current version of layerUMAP is implemented in python and can be found at https://github.com/jingry/autoBioSeqpy/blob/2.0/tool/layerUMAP.py. All deep learning models and commands for the case studies were integrated as a compressed file at https://github.com/jingry/autoBioSeqpy/blob/2.0/examples/layerUMAP.zip.
- Any additional information required to reanalyze the data reported in this paper is available from the lead contact upon request.

METHOD DETAILS

Overview of layerUMAP
As the main plugin of the autoBioSeqpy toolbox, layerUMAP enables visual analysis of trained DL models using a one-line command. It provides several parameters and options that can be easily set by users, including the following:

- –paraFile: a path parameter containing information about the trained DL models;
- –outFigFolder: a path parameter specifying the output folder in which the result file named UMAP.pdf will be saved;
- –n_neighbors: a UMAP parameter controlling how UMAP balances local versus global structure in the data;
• –min_dist: a UMAP parameter controlling how tightly UMAP is allowed to pack points together;
• –metric: a UMAP parameter controlling how distance is computed in the ambient space of the input data;
• –grid_n_neighbors and –grid_min_dist: two parameters for performing grid searches for the –n_neighbors and –min_dist parameters, respectively. The values of the two parameters are passed to arrange () in NumPy (used as ‘start stop stepSize’). If ‘–grid_min_dist’ or ‘–grid_n_neighbors’ is used, ‘–min_dist’ or ‘–n_neighbors’ will be ignored, respectively;
• –theme: a parameter specifying the colour, background and theme for plotting;
• –figWidth and –figHeight: two parameters specifying the size of the output figures;
• –dimensionReduceMethod: a parameter for reducing the high-dimensional outputs to two dimensions, with possible values of ‘max’, ‘min’, ‘avg’ or ‘flatten’;
• –interactive: a parameter specifying a layer name or layer index for plotting.

To make the processes of layer selection and dimensionality reduction more user-friendly, an interactive option is provided in layerUMAP by setting ‘–interactive 1’ in the command. When the interactive option is turned on, users are asked to select which layer to use as the output layer by specifying the name or index of the layer in the list. If the output dimensionality is higher than the value of 2 required by UMAP, three options are provided for reducing the dimensionality: 1) selecting a 2-D slice from the N-D tensor, 2) splitting the N-D tensor into 2-D slices and plotting all slices, and 3) changing the N-D tensor to 2-D using the maximal, minimal, average, or flatten method. Users can specify the dimensions or axes to be preserved or reduced, except for the first dimension, which represents the number of instances. The visual output is generated in PDF format, and the number of pages will be greater than 1, depending on the parameters.

Dataset for case study 1

A large number of pathogenic and symbiotic bacteria have evolved specialized secretion systems to deliver bacterial proteins into eukaryotic cells to ensure their survival and replication. To date, nine secretion systems have been found in Gram-negative bacteria and named from the type I (T1SS) to the type IX secretion system (T9SS) according to their outer membrane secretion mechanisms. Correspondingly, proteins released through T1SS are called type I secreted proteins (T1SPs), and other types of proteins are named analogously. These secreted proteins play different roles in invaded eukaryotic cells and cause various diseases; therefore, it is crucial to explore their relationships and differences, which is beneficial for both disease pathogenesis and drug development.

The dataset used in this case was obtained from our previous work. Each protein was categorized into one of six subsets, namely, from T1SP to T7SP, with the exception of T6SP. In more detail, we first collected information concerning protein or gene IDs and corresponding secretory types through a comprehensive literature search and then manually extracted protein sequences from three data sources, namely, Swiss-Prot, TrEMBL and RefSeq. To ensure an accurate collection, only reviewed entries (that is, from Swiss-Prot) were used, and ‘hypothetical’, ‘probable’, ‘by similarity’ or ‘predicted’ proteins were not included. We discarded protein sequences shorter than 50 AAs and further filtered duplicate or homologous proteins with a sequence identity threshold of 25% using the BLASTCLUST program. With this procedure, we ended up with 839 proteins, including 667 training proteins and 172 test proteins, whose secretory types are distributed as follows: T1SP (112, 25), T2SP (99, 29), T3SP (182, 28), T4SP (62, 22), T5SP (164, 35) and T7SP (48, 33).

Deep learning architectures for case study 1

We selected and trained five representative DL models constructed with different network architectures, including a convolutional neural network (CNN) and four recurrent neural network (RNN)-based variants, namely, bidirectional long short-term memory (BiLSTM), bidirectional gated recurrent unit (BiGRU), CNN-BiLSTM and CNN-BiGRU. We searched for the best model from each proposed architecture and explored different possibilities for developing better models by trying to tune the hyperparameters and optimize the architectures. It is generally accepted that CNNs perform well at capturing local patterns in sequences, whereas RNNs can capture long-distance dependencies in sequential data. To fairly evaluate the abilities of CNNs and RNNs to learn sequence patterns within Gram-negative bacterial secreted
proteins, we changed only the convolutional and recurrent layers in each proposed architecture while keeping the other types of network layers consistent. All models consume raw sequences as input and calculate the probability of a protein of interest being one of the six secretory types. More precisely, the input to each model is a sequence of one-hot encoded amino acids, where each residue is transformed into a 20-dimensional vector, and the output of each model consists of six scores that sum to one.

The CNN architecture consists of two convolutional layers, a pooling layer, a dropout layer, an activation layer and two fully connected layers. Both convolutional layers first perform 1D convolution operations with 300 filters of kernel length 5. Each convolutional layer applies a ReLU activation function to introduce non-linearities. The third pooling layer computes the global maximum value in each feature map, increasing the translational invariance and reducing the number of parameters. The pooled output is then passed through a fully connected layer containing 650 nodes with a 0.5× dropout and finally through another fully connected layer with softmax activation, producing 6 secretory type predictions. We used the BiLSTM and BiGRU as RNN representation learners considering their successful applications to similar tasks in biological sequences. Specifically, the two RNN architectures selected for training were a three-layer stacked BiLSTM with 128-dimensional hidden state vectors per layer and a three-layer stacked BiGRU, also with 128 dimensions per layer, both followed by GlobalMaxPooling layers and two fully connected layers, which were the same settings as in the CNN architecture. The principal network architectures of CNN-BiLSTM and CNN-BiGRU were adapted from the above three architectures. Two convolutional layers (filters 300, 300; kernel widths 5, 5) with ReLU activation, three BiLSTM or BiGRU layers (hidden state vectors 128, 128, 128), a GlobalMaxPooling layer, a fully connected layer (650 hidden units), dropout (rate = 0.5) and ReLU activation were used, followed by a fully connected layer with a softmax function to output individual probabilities for each label.

All models were trained for 50 epochs with a batch size of 40 and a learning rate of 0.001 on an NVIDIA GTX 1060 GPU to enable parallel calculation of the gradient. For general user applications, the CPU backend was found to be sufficient. The categorical cross-entropy loss between the predicted labels and the true observed labels was minimized using the Adam optimizer during training. As is conventional, the test set was used for hyperparameter tuning and model architecture optimization. The length of the input sequence was limited to 2000 amino acids, this limit was chosen to fit the longest sequence in the dataset.

Datasets for case study 2

The recent maturation of single-cell RNA sequencing (scRNA-seq) technology has revolutionized our ability to identify and characterize cell types, states, lineages and circuitries. Rapid and continuous technological advances in scRNA-seq now enable routine profiling of millions of cells. Such large and growing data have opened the way for substantial scientific discoveries and have driven a global project, namely, the Human Cell Atlas (HCA), which aims to build a comprehensive reference map of all human cells. Despite the unprecedented power of scRNA-seq, it remains a computationally challenging task to define and identify cell populations in a given dataset due to the high dimensionality and inherited high level of technical noise of scRNA-seq data. Marker gene-based manual annotation after unsupervised clustering is still the standard practice for solving this task. However, the manual annotation process is cumbersome, time-consuming and potentially inaccurate. To address this pressing challenge, a growing number of supervised classification methods have been developed for cell type assignment by learning the identities of annotated cells from a reference atlas.

Abdelaal et al. benchmarked 22 classification methods for the automatic assignment of cell types using 27 publicly available scRNA-seq datasets of different sizes, technologies, species, and complexities. In this case study, we selected three benchmark datasets from the study of to develop a novel deep learning model for cell type classification. The CellBench dataset consists of 4,373 cells from five human lung cancer cell lines assayed by two different technologies (10X chromium and CEL-Seq2). The five cell populations are A549 (1,410), H1975 (517), H2228 (828), H838 (986), and HCC827 (632). The human pancreatic dataset comprises four individual scRNA-seq datasets (Baron Human, Muraro, Segerstolpe, and Xin) with a total of 10,150 cells. We selected the four major endocrine pancreatic cell types, namely, alpha (4,865), beta (3,702), delta (950) and gamma (633), across all four pancreatic datasets. The Zheng sorted dataset contains 20,000 cells in total and was obtained by downsampling 2,000 cells for each population from the full datasets. The 10 representative PBMC-sorted populations selected from the 10X Genomics website are CD14+ monocytes, CD19+ B cells, CD34+ cells, CD4+ helper T cells, CD4+/CD25+ regulatory T cells,
CD4+/CD45RA+/CD25-naive T cells, CD4+/CD45RO + memory T cells, CD56+ natural killer cells, CD8+ cytotoxic T cells, and CD8+/CD45RA + naive cytotoxic T cells. All datasets were processed, aligned and merged using the following procedure. Uncertain cells annotated as ‘doublets’, ‘debris’, ‘unlabelled cell’, ‘not applicable’, and ‘unclassified cell’, as well as cell types with fewer than 10 cells across the entire dataset, were filtered out. Genes expressing zero counts in all cells and cells expressing a median absolute deviation (MAD) of less than 3 were removed. In addition, to correct batch effects and technical variations between different protocols, datasets were aligned using MNN\textsuperscript{43} from the scran R package.

**Deep learning architectures for case study 2**

With reference to the ResNet architecture,\textsuperscript{44} we developed a residual fully connected network for determining cell identities directly using scRNA-seq data. The input gene expression data are first fed into two fully connected layers with 5120 and 256 neurons. Afterwards, the output from these layers is concatenated and input into a residual block module, and this process is repeated 10 times. In the residual block, three fully connected layers (with 256, 1024, and 256 hidden nodes) and layer normalization (LN) layer share the weights in each inner block iteration, where the three fully connected layers constitute a bottleneck structure and the LN layer is used to restrict the features to the same average and variance. Following the residual block module is a fully connected layer with 128 nodes and the last output layer with a softmax activation layer. We used the ReLU activation function and a dropout rate of 0.15 in all fully connected layers to avoid overfitting. Detailed pseudocode for the residual fully connected network is given in Algorithm 1.

```
Algorithm 1. Pseudocode of ResNet
Function ResidualFullConnModel(Xin)
1: X = Dropout(Relu(FC256(Dropout(Relu(FC5120(Xin))))))
2: for i from 1 to 10:
3:   Xtmp = Dropout(Relu(FC256(LayerNorm(X)))))
4:   Xtmp = Dropout(Relu(FC1024(Xtmp)))
5:   X+ = Relu(FC256(Xtmp))
6: end for
7: Out = Softmax(FC4(Dropout(FC128(X)))))
8: Return Out
```

Each input scRNA-seq dataset was divided into a testing set (10%) and a development set (90%). From the development set, a validation set of the same size as the testing set was selected, and the remaining cells were reserved for training. The model was trained to minimize a categorical cross-entropy loss using the Adam optimizer with a batch size of 128. The learning rate of the optimizer was set to 0.0005, and training was limited to 20 epochs.

**QUANTIFICATION AND STATISTICAL ANALYSIS**

Confusion matrix and total accuracy of deep learning model were provided by autoBioSeqpy. Heat maps, bar charts, 3D bar charts, line plots and matrix bubbles were generated using the Hiplot software.\textsuperscript{45} Generation of UMAPs were performed using the command-line commands detailed above.