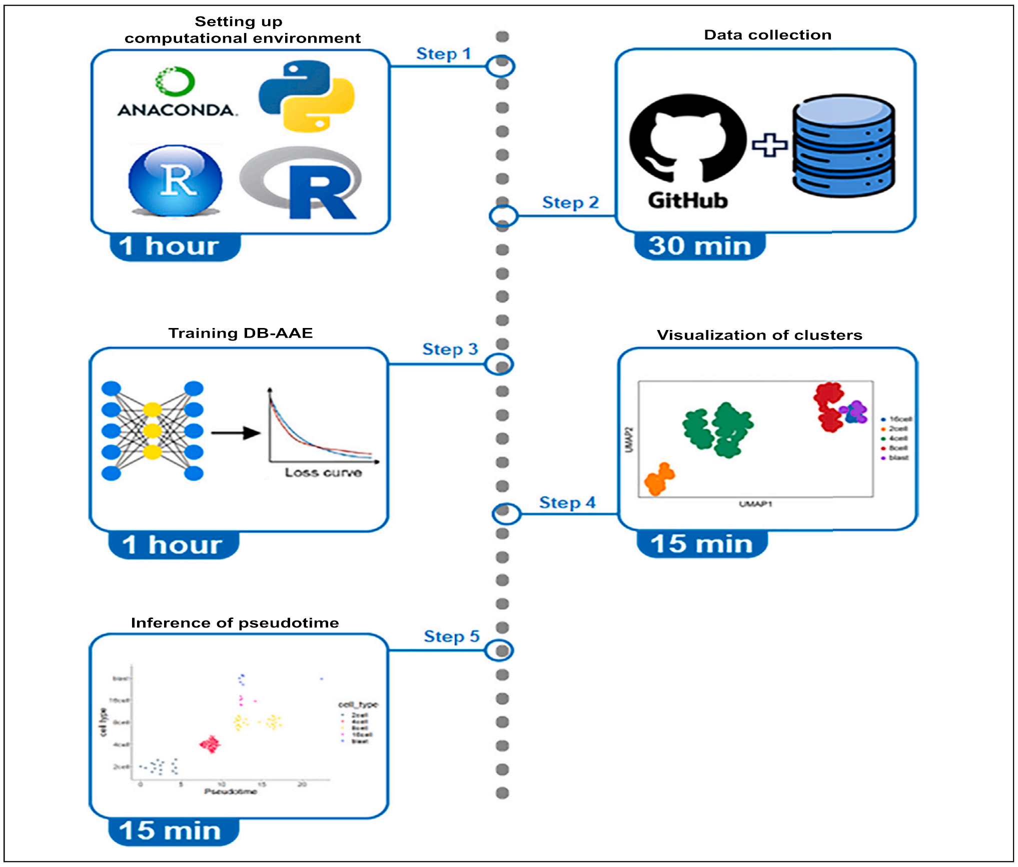
Protocol

A deep learning framework for denoising and ordering scRNA-seq data using adversarial autoencoder with dynamic batching



Single-cell RNA sequencing (scRNA-seq) provides high resolution of cell-to-cell variation in gene expression and offers insights into cell heterogeneity, differentiating dynamics, and disease mechanisms. However, technical challenges such as low capture rates and dropout events can introduce noise in data analysis. Here, we present a deep learning framework, called the dynamic batching adversarial autoencoder (DB-AAE), for denoising scRNA-seq datasets. First, we describe steps to set up the computing environment, training, and tuning. Then, we depict the visualization of the denoising results.

Publisher's note: Undertaking any experimental protocol requires adherence to local institutional guidelines for laboratory safety and ethics.

Kyung Dae Ko,
Vittorio Sartorelli

vittorio.sartorelli@nih.gov

Highlights

Steps for denoising scRNA-seq using dynamic batching adversarial autoencoder (DB-AAE)

Instructions to train DB-AAE and visualize denoising results

Guidance on inferring pseudotime from the denoising results

Ko & Sartorelli, STAR Protocols 5, 103067
June 21, 2024 Published by Elsevier Inc.
https://doi.org/10.1016/j.xpro.2024.103067

CellPress
OPEN ACCESS

Protocol

A deep learning framework for denoising and ordering scRNA-seq data using adversarial autoencoder with dynamic batching

Kyung Dae Ko[1,2] and Vittorio Sartorelli[1,3,*]

[1]Laboratory of Muscle Stem Cells and Gene Regulation, NIAMS, NIH, Bethesda, MD, USA

[2]Technical contact

[3]Lead contact

*Correspondence: vittorio.sartorelli@nih.gov

https://doi.org/10.1016/j.xpro.2024.103067

SUMMARY

Single-cell RNA sequencing (scRNA-seq) provides high resolution of cell-to-cell variation in gene expression and offers insights into cell heterogeneity, differentiating dynamics, and disease mechanisms. However, technical challenges such as low capture rates and dropout events can introduce noise in data analysis. Here, we present a deep learning framework, called the dynamic batching adversarial autoencoder (DB-AAE), for denoising scRNA-seq datasets. First, we describe steps to set up the computing environment, training, and tuning. Then, we depict the visualization of the denoising results.

For complete details on the use and execution of this protocol, please refer to Ko et al.[1]

BEFORE YOU BEGIN

Deep learning techniques, particularly autoencoders, have been developed to address noise reduction and missing value imputation challenges in single-cell RNA sequencing (scRNA-seq) data analysis. However, these methods can exhibit sensitivity to sparse data and batch effects. To improve denoising performance and alleviate information loss during analysis, we propose an approach called the dynamic batching adversarial autoencoder (DB-AAE). This model utilizes a competitive architecture that directly captures optimal features from input data, bypassing the need for traditional statistical models.

The DB-AAE comprises three crucial components: encoder, decoder, and adversary modules (Figure 1). This protocol provides comprehensive step-by-step instructions to denoise scRNA-seq data using the DB-AAE framework, suitable for both beginners and seasoned professionals.

For complete details on the use and execution of this protocol, please refer to Ko et al.[1]

Hardware preparation

The whole framework can run on a MacOS or Window operation system. Even though the RAM requirement depends on the number of cells to be analyzed, 16 GB RAM should be sufficient for an initial analysis.

Software preparation

This framework is based on the Microsoft Windows operating system but it can be run on MacOS. To build a functional Python environment, this protocol uses Anaconda v2.5.1 as Anaconda provides various integrated development environment (IDE)s for code development and management.

This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

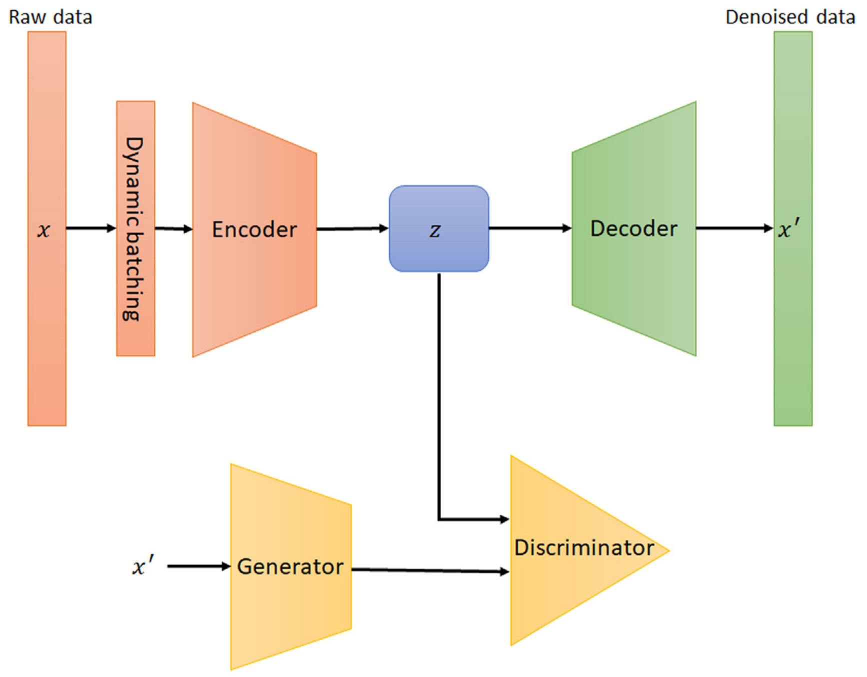


Figure 1. Structure of the dynamic batching adversarial autoencoder (DB-AAE)

Among the IDEs in the Anaconda, JupyterLab IDE v.3.5.3 was selected as it easily visualizes the results through the framework. For additional visualization, R studio v.2023.06.2 + 561 with R v.4.3 was used.

Setting up the computational environment

Timing: 1 h

1. Install Anaconda.
   a. Download Anaconda from "https://repo.anaconda.com/archive/"
   b. Install the Anaconda Navigator following the default settings.
   c. Launch the Anaconda Navigator.
2. Set up Virtual environment.
   a. Create a new environment called to "db_aae" from Environments (Figures 2A and 2B).
   b. Install JupyterLab from Home after selecting "db_aae" environment (Figure 2C).
   c. Launch JupyterLab from Home (Figure 2D).
3. Install python packages.
   a. Launch a terminal in JupyterLab (Figure 3A).
   b. Run the following command in the terminal to install DB-AAE (Figure 3B).

```
> pip install dbaae
```

4. Install R and R Studio.
   a. Download R from "https://cloud.r-project.org/" and install it.
   b. Download R-studio from "https://posit.co/download/rstudio-desktop/" and install it.
5. Install R packages.
   a. Install scater and slingshot.

```
> if (!require("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install(c("scater","slingshot"))
```

b. Install Polychrome.

```
> install.packages(``Polychrome'')
```

c. Install zellkonverter.

```
> if (!require("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install("zellkonverter")
```

d. Install scater.

```
> if (!require("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install("scater")
```

Data collection

Timing: 30 min

In this section, we collect raw scRNA-seq data from a public archive (https://hemberg-lab.github.io/scRNA.seq.datasets/mouse/edev/) and convers the data to a Hierarchical Data Format (HDF-.h5ad) for python.

6. Import R libraries.

```
library("scater")
library(zellkonverter)
```

7. Convert a dataset to H5ad format and save the dataset.

```
goolam<-readRDS("../pre_process_data/goolam.rds")
rowData(goolam)
colData(goolam)
writeH5AD(goolam, file = "goolam.h5ad")
```

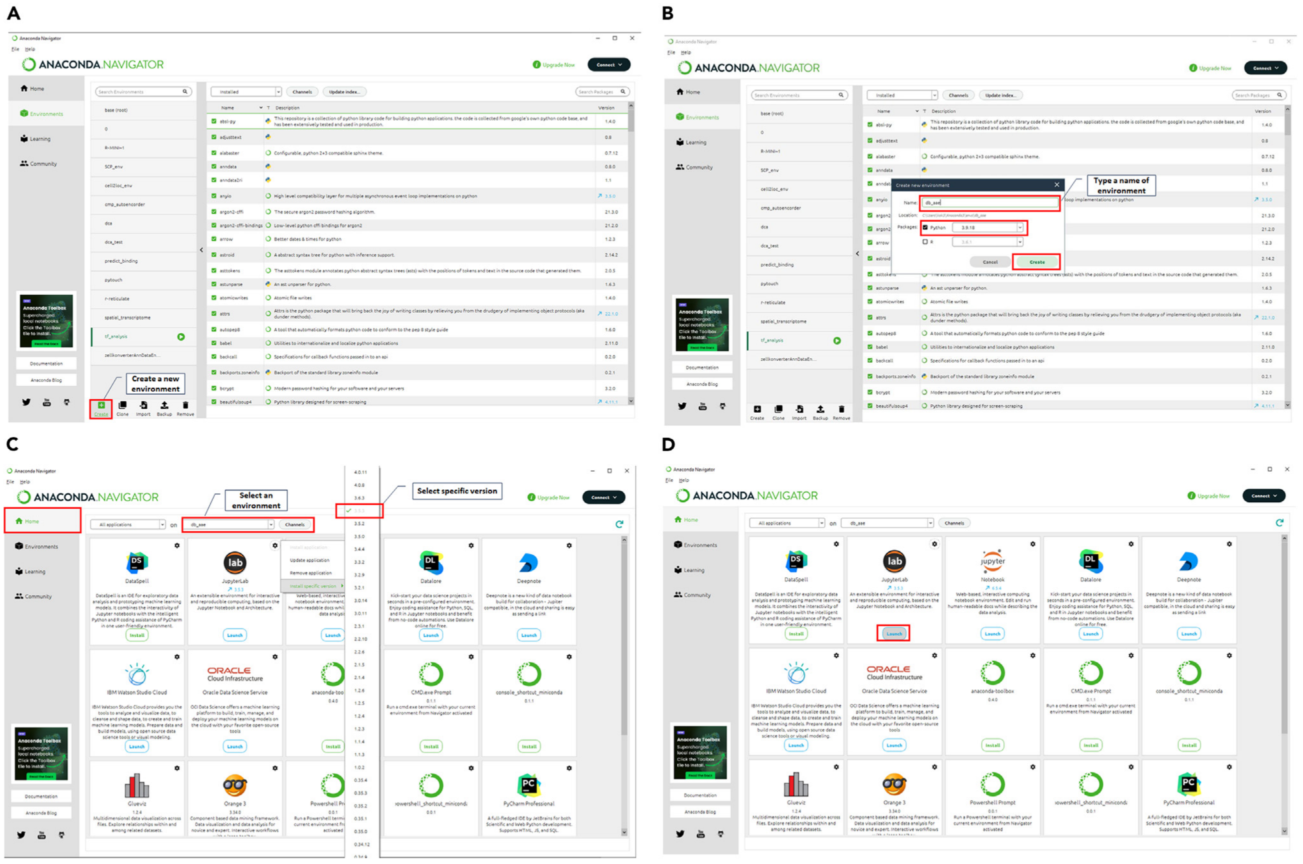


Figure 2. Anaconda Navigator
(A) "Environments" page illustration.
(B) Illustration to create a new conda environment.
(C) "Home" page illustration to install JupyterLab.
(D) Illustration to launch JupyterLab.

Note: Single-cell datasets analyzed in this protocol were preprocessed and deposited into Google Drive. The link is represented in GitHub (https://github.com/LMSCGR/DB-AAE).

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Deposited data		
goolam	Goolam et al.[2]	https://drive.google.com/drive/u/1/folders/1DwRpIEStBTQ7XfAel7Qq1uJeCRZyUj9U
baron	Baron et al.[3]	https://drive.google.com/drive/u/1/folders/1DwRpIEStBTQ7XfAel7Qq1uJeCRZyUj9U
Software and algorithms		
R v.4.3	The R Project for Statistical Computing	https://www.r-project.org/
RStudio server (v.2023.06.2+561)	RStudio Team, 2022	https://posit.co/
Python v.3.8	Python Software Foundation	https://cran.r-project.org/web/packages/Seurat/index.html
scanpy v.1.9.1	Wolf et al.[4]	https://github.com/scverse/scanpy
seaborn v.0.12.2	Waskom[5]	https://github.com/mwaskom/seaborn
pandas v.1.5.3	McKinney[6]	https://github.com/pandas-dev/pandas
matplotlib v.3.6.3	Hunter[7]	https://github.com/matplotlib/matplotlib
numpy v.1.23.5	Harris et al.[8]	https://github.com/numpy/numpy

(*Continued on next page*)

Continued

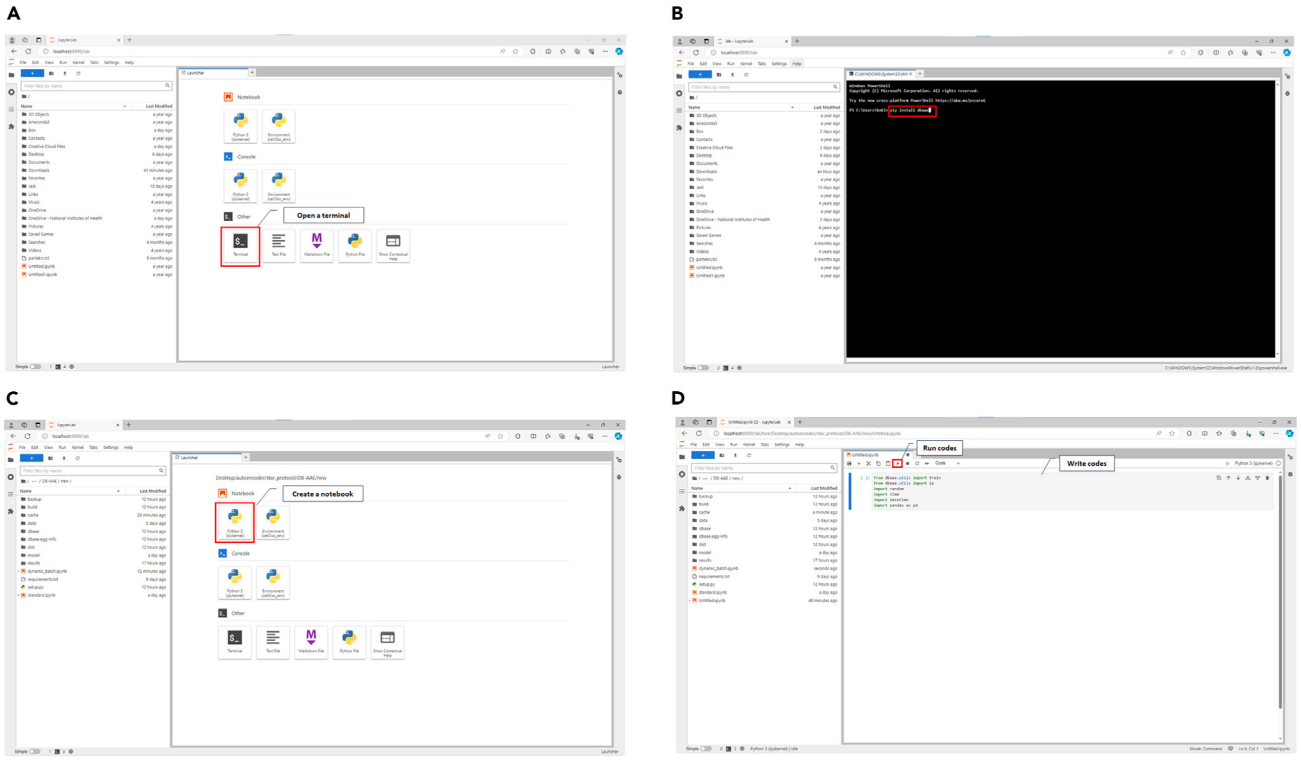
REAGENT or RESOURCE	SOURCE	IDENTIFIER
TensorFlow v.2.11.0	TensorFlow team, 2015	https://www.tensorflow.org/
keras-tuner v.1.4.5		https://github.com/keras-team/keras-tuner
keras v.2.11.0	Keras team, 2015	https://github.com/fchollet/keras
zellkonverter v.1.12.1		https://github.com/theislab/zellkonverter
Polychrome v.1.5.1		https://cran.r-project.org/web/packages/Polychrome/index.html
Slingshot v.2.10.0	Street et al.[9]	https://github.com/kstreet13/slingshot
scater v.1.28.0	McCarthy et al.[10]	http://bioconductor.org/packages/scater/

STEP-BY-STEP METHOD DETAILS

In this section, we describe essential steps to train DB-AAE (Dynamic batching adversarial auto encoder).

Step 1: Training DB-AAE

⏲ Timing: 1 h



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\kok3> pip install dbaae
```

```
from dbaae.utils import train
from dbaae.utils import io
import random
import time
import datetime
import pandas as pd
```

Figure 3. Setting up the computational environment
(A) JupyterLab illustration.
(B) Illustration to run a terminal for installation of DB-AAE package.
(C) Illustration to launch a Jupyter notebook.
(D) Illustration to run python codes in a Jupyter notebook.

1. Create jupyter notebook (Figure 3C).

2. Import python package.

```
from dbaae.utils import train

from dbaae.utils import io

import random

import time

import datetime

import pandas as pd
```

3. Load a dataset.

```
random.seed(1234)

input_path="./data/goolam.h5ad"

genenum=5000

latent_dim=512

adata=io.read_dataset(input_path,highly_genes=highly_genes)
```

4. Setup hyper parameters.

```
input_layer=genenum

hidden_layer1=1024

hidden_layer2=512

latent_layer=latent_dim

batch_size = 32

n_epochs = 25

X_train = pd.DataFrame(adata.X)

X_train =X_train.to_numpy()

activation='relu'

optimizer="RMSprop"

learningrate=2e-05

start_time = time.time()
```

Note: To use a specific dataset, the "input_path" can be changed to the location with the specific filename. If the accuracy after step 5 is not high, "optimizer" and "learningrate" can be changed following guide from keras website (https://keras.io/api/optimizers/).

5. Train DB-AAE.

a. Train with static batching (or).

```
adversarial_autoencoder=train.train(X_train,input_layer,hidden_layer1,hidden_layer2,latent_layer,batch_size,
n_epochs,learningrate,activation,optimizer)

adversarial_autoencoder.summary()
```

b. Train with dynamic batching.

```
from dbaae.utils import dynamic

import keras_tuner as kt

import tensorflow as tf

tuner = kt.Hyperband(dynamic.AEHyperModel(X_train,input_layer,hidden_layer1,hidden_layer2,latent_layer,
n_epochs,learningrate,activation,optimizer),

        objective = 'val_accuracy',

        max_epochs = n_epochs,

        executions_per_trial = 1,

        overwrite = True,

          directory = './cache',

          max_consecutive_failed_trials=5,

        factor = 3)

stop_early = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', mode='max', baseline=100)

tuner.search(X_train, X_train, epochs = n_epochs, validation_split = 0.1, callbacks=[stop_early])

best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]

print(f"""batch_size : {best_hps.get('batch_size')}""")

batch_size = best_hps.get('batch_size')

random.seed(1234)

adversarial_autoencoder=train.train(X_train,input_layer,hidden_layer1,hidden_layer2,latent_layer,batch_size,
n_epochs,learningrate,activation,optimizer)

adversarial_autoencoder.summary()
```

⚠ CRITICAL: Step 5-a describes static batching and step-b is dynamic batching. In step 5-b, to use dynamic batching for training, "n_echochs" can be changed to small number to reduce training time since the training time of dynamic batching is 2–3 time slower than that of static batching. Follow guidelines from keras website (https://keras.io/api/keras_tuner/tuners/hyperband/).

6. Save the trained model.

```
rlt_dir='./models'

current_time = datetime.datetime.now()

adversarial_autoencoder.save(rlt_dir+'{}'.format("DB_AAE")+'_'+-
str(current_time.year)+'_'+str(current_time.month)+'_'+
str(current_time.day)+'_'+str(current_time.hour)+'_'+str(current_time.minute)+'.h5')
```

Step 2: Visualize the clusters of the denoised scRNA-seq dataset

Timing: 15 min

7. Import the trained model.

```
import keras
db_aae = keras.models.load_model('./model/DB_AAE_2024_2_20_10_31.h5')
```

8. Load original data.

```
random.seed(1234)
input_path="./data/goolam.h5ad"
genenum=5000
latent_dim=512
adata=io.read_dataset(input_path,highly_genes=genenum)
```

9. Reconstruct the data.

```
X_train = pd.DataFrame(adata.X)
X_train =X_train.to_numpy()
cellinfo=pd.DataFrame(adata.obs['cell_type1'])
geneinfo=pd.DataFrame(adata.var['feature_symbol'])
counts_aae = db_aae.predict(X_train)
import scanpy as sc
adata_aae = sc.AnnData(counts_aae[0],obs=cellinfo,var=geneinfo)
adata_aae.write("./results/DB_AAE_2024_2_20_10_31.h5ad")
```

10. Preprocess the data for visualization of clusters using SCANPY.

```
sc.pp.normalize_per_cell(adata_aae)
sc.pp.log1p(adata_aae)
sc.pp.pca(adata_aae)
sc.pl.pca_variance_ratio(adata_aae)
sc.pp.neighbors(adata_aae, n_neighbors=60, n_pcs=10)
```

11. Visualize the clusters (Figure 4A).

```
sc.tl.umap(adata_aae)
sc.pl.umap(adata_aae, color='cell_type1',title='Reconstructed data')
```

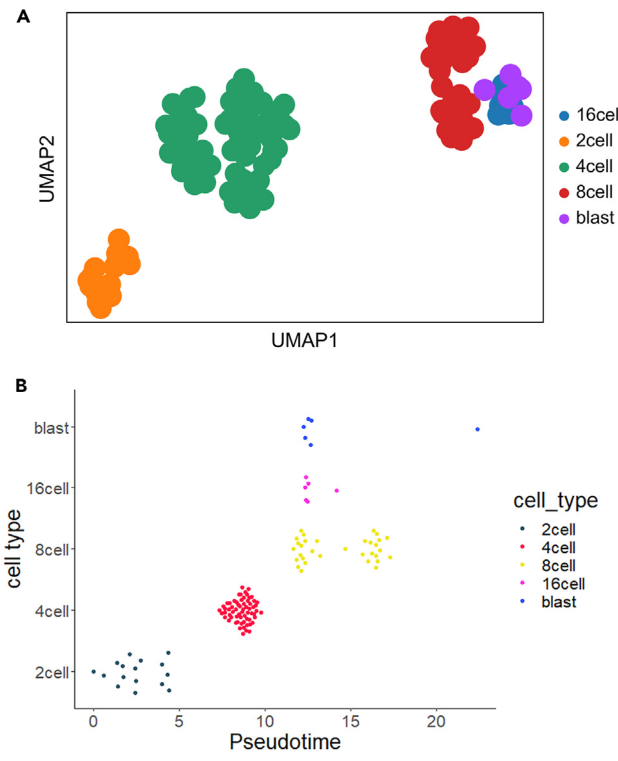


Figure 4. Visualization of a denoised scRNA-seq dataset
(A) UMAP plot of a denoised scRNA-seq dataset.
(B) Plot to infer pseudo time of a denoised scRNA-seq dataset.

Note: "n_neighbors" controls how UMAP balances local versus global structure in the data and "n_pcs" decides the number of PCs involved in clustering. Their numbers are decided based on "pca_variance_ratio" plot in step 10. After the value of n_neighbors that makes "pca_variance_ratio" plot smooth is selected, the point that starts to smoothen and flatten out in the plot is chosen for the value of n_pcs.

Step 3: Infer pseudotime of the denoised scRNA-seq dataset

Timing: 15 min

12. Infer pseudotime from the denoised dataset using R.
    a. Import R libraries.

```
library("slingshot")

library("scater")

library(zellkonverter)

library(ggplot2)

library(ggbeeswarm)

library(ggthemes)

library(Polychrome)
```

b. Load a dataset.

```
aae_goolam_sce<-readH5AD("./results/DB_AAE_2024_2_20_10_31.h5ad")

logcounts(aae_goolam_sce)<-assay(aae_goolam_sce)
```

c. Preprocess a dataset for the visualization of pseudo time.

```
aae_goolam_sce@colData$cell_type1<-ordered(aae_goolam_sce@colData$cell_type1,
levels = c("2cell", "4cell","8cell","16cell","blast"))

cell.stages= c("2cell", "4cell","8cell","16cell","blast")

label=ordered(aae_goolam_sce$cell_type1, levels = c("2cell",
"4cell","8cell","16cell","blast"))

set.seed(723451)

my_color <- createPalette(10, c("#010101", "#ff0000"), M=1000)

names(my_color) <- unique(as.character(aae_goolam_sce$cell_type1))

aae_goolam_sce <- runPCA(aae_goolam_sce, ncomponents = 20)

aae_goolam_sce <- runUMAP(aae_goolam_sce)

aae_goolam_sce <- slingshot(aae_goolam_sce,reducedDim = "PCA")

slingshot_df <- data.frame(slingPseudotime_1=aae_goolam_sce@colData$slingPseudotime_1,
cell_type=aae_goolam_sce@colData$cell_type1)

slingshot_df$cell_type <- ordered(slingshot_df$cell_type, levels = c("2cell",
"4cell","8cell","16cell","blast"))

set.seed(723451) # for reproducibility

my_color <- createPalette(10, c("#010101", "#ff0000"), M=1000)

names(my_color) <- unique(as.character(aae_goolam_sce$cell_type1))

aae_goolam_sce <- runPCA(aae_goolam_sce, ncomponents = 20)

aae_goolam_sce <- runUMAP(aae_goolam_sce)

aae_goolam_sce <- slingshot(aae_goolam_sce,reducedDim = "PCA")

slingshot_df <- data.frame(slingPseudotime_1=aae_goolam_sce@colData$slingPseudotime_1,
cell_type=aae_goolam_sce@colData$cell_type1)

slingshot_df$cell_type <- ordered(slingshot_df$cell_type, levels = c("2cell", "4cell",
"8cell","16cell","blast"))
```

d. Visualize the pseudotime (Figure 4B).

```
library(ggplot2)

library(ggbeeswarm)

library(ggthemes)

ggplot(slingshot_df, aes(x = slingPseudotime_1, y = cell_type,

           colour = cell_type)) +

  geom_quasirandom(groupOnX = FALSE) + scale_color_tableau() + theme_classic() +
  xlab("Pseudotime") + ylab("cell type") +

  scale_colour_manual(values = my_color)+theme(text = element_text(size = 20))
```

EXPECTED OUTCOMES

This protocol outlines a generative framework named DB-AAE designed to tackle the inherent challenges related to denoising and imputation in single-cell RNA sequencing (scRNA-seq) data. By employing a dynamic batching procedure in step 1 during the training of DB-AAE, an optimized model with an ideal batch size can be constructed. The trained model can generate a denoised scRNA-seq dataset in h5ad format. Subsequently, in step 2, the denoised dataset can be visualized, and its pseudo-time can be inferred.

LIMITATIONS

As the implementation of DB-AAE relies on a deep-learning model, this framework faces limitations in offering detailed insights into the specific acquisition and utilization of features or gene expression patterns by the model. This constraint arises from the intricate nature of deep learning models, characterized by numerous layers with complex interactions among nodes. Consequently, the internal workings or processes of the model are challenging to comprehend or interpret. Moreover, the performance of the DB-AAE framework is highly sensitive to hyperparameters, necessitating thorough hyperparameter tuning for ensuring the method's stability and robustness.

TROUBLESHOOTING

Problem 1

HTTP error when installing Python packages (related to Step 3).

Potential solution

This is due to the poor quality of internet connection to the package sources. A possible solution to solve this problem is to check internet connection. If the connection is recovered, repeat steps in Figures 2C, 2D, and 3A.

Problem 2

IPython Console reports an error of ''No module named XX'' when executing the codes (related to Step 3).

Potential solution

- This is because the correct package version is not installed. Open the terminal (Figure 3A) and check the installed package with this command.

```
conda list
```

- Check python packages in GitHub (https://github.com/LMSCGR/DB-AAE) and reinstall the packages with this command in the terminal.

```
pip install package name == version
```

Problem 3

R packages cannot be loaded by "library" command (related to Step 6).

Potential solution

Run the codes in R environment below.

```
p <- installed.packages()

colnames(p)
```

If the packages cannot be found after running the codes, visit the Bioconductor website (https://www.bioconductor.org/), search a package, and follow guidelines. If the package cannot be found in Bioconductor, run **install.packages("package_name")** in R environment.

Problem 4

Data files cannot be loaded (related to Step 7).

Potential solution

Check whether the files are in the folder. If they are, check their name.

Problem 5

Plots cannot be drawn (related to Step 11 and Step 12f).

Potential solution

Run the codes in R environment below.

```
gg2 <- try(find.package("ggplot2"), silent = TRUE)

gg2
```

If the packages cannot be found after running the codes, run **install.packages("ggplot2")** in R environment.

RESOURCE AVAILABILITY

Lead contact

Further information and requests for resources and reagents should be directed to and will be fulfilled by the lead contact, Vittorio Sartorelli (vittorio.sartorelli@nih.gov).

Technical contact

Technical questions on executing this protocol should be directed to and will be answered by the technical contact, Kyung Dae Ko (kyungdae.ko@nih.gov).

Materials availability

This study did not generate unique reagents.

Data and code availability

Original data and codes have been deposited to Zenodo: https://doi.org/10.5281/zenodo.10478925.

ACKNOWLEDGMENTS

We thank members of the Sartorelli's laboratory for insightful comments. Funding has been provided by NIH grants AR041126 and AR041164.

AUTHOR CONTRIBUTIONS

K.D.K. analyzed and interpreted data and drafted the manuscript. V.S. edited the manuscript and supervised the project.

DECLARATION OF INTERESTS

The authors declare no competing interests.

REFERENCES

1. Ko, K.D., and Sartorelli, V. (2024). A deep learning adversarial autoencoder with dynamic batching displays high performance in denoising and ordering scRNA-seq data. iScience *27*, 109027.

2. Goolam, M., Scialdone, A., Graham, S.J.L., Macaulay, I.C., Jedrusik, A., Hupalowska, A., Voet, T., Marioni, J.C., and Zernicka-Goetz, M. (2016). Heterogeneity in Oct4 and Sox2 Targets Biases Cell Fate in 4-Cell Mouse Embryos. Cell *165*, 61–74.

3. Baron, M., Veres, A., Wolock, S.L., Faust, A.L., Gaujoux, R., Vetere, A., Ryu, J.H., Wagner, B.K., Shen-Orr, S.S., Klein, A.M., et al. (2016). A Single-Cell Transcriptomic Map of the Human and Mouse Pancreas Reveals Inter- and Intra-cell Population Structure. Cell Syst. *3*, 346–360.e4.

4. Wolf, F.A., Angerer, P., and Theis, F.J. (2018). SCANPY: large-scale single-cell gene expression data analysis. Genome Biol. *19*, 15.

5. Waskom, M. (2021). seaborn: statistical data visualization. J. Open Source Softw. *6*, 3021.

6. McKinney, W. (2010). Data Structures for Statistical Computing in Python. In Proceedings of the 9th Python in Science Conference, *445*.

7. Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., et al. (2020). Array programming with NumPy. Nature *585*, 357–362.

8. Hunter, J.D. (2007). Matplotlib: A 2D Graphics Environment. Comput. Sci. Eng. *9*, 90–95.

9. Street, K., Risso, D., Fletcher, R.B., Das, D., Ngai, J., Yosef, N., Purdom, E., and Dudoit, S. (2018). Slingshot: cell lineage and pseudotime inference for single-cell transcriptomics. BMC Genom. *19*, 477.

10. McCarthy, D.J., Campbell, K.R., Lun, A.T.L., and Wills, Q.F. (2017). Scater: pre-processing, quality control, normalization and visualization of single-cell RNA-seq data in R. Bioinformatics *33*, 1179–1186.