SCALEX

Release 0.2.0

Lei Xiong

Oct 12, 2021
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Tutorial</td>
<td>1.1 Integration</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1.2 Projection</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1.3 Label transfer</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1.4 Integration scATAC-seq data</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>1.5 Integration cross-modality data</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>1.6 Spatial data (To be updated)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>1.7 Examples</td>
<td>8</td>
</tr>
<tr>
<td>2 Installation</td>
<td>2.1 PyPI install</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>2.2 Pytorch</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>2.3 Troubleshooting</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>2.4 Anaconda</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>2.5 Installing Miniconda</td>
<td>45</td>
</tr>
<tr>
<td>3 Usage</td>
<td>3.1 Command line</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>3.2 API function</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>3.3 AnnData</td>
<td>48</td>
</tr>
<tr>
<td>4 API</td>
<td>4.1 Function</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>4.2 Data</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>4.3 Net</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>4.4 Plot</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>4.5 Metric</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>4.6 Logger</td>
<td>73</td>
</tr>
<tr>
<td>5 News</td>
<td></td>
<td>75</td>
</tr>
<tr>
<td>6 Release</td>
<td>6.1 Version 2.0</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>6.2 Version 1.0</td>
<td>77</td>
</tr>
<tr>
<td>7 News</td>
<td>7.1 Indices and tables</td>
<td>79</td>
</tr>
</tbody>
</table>

Python Module Index | 81 |
Index | 83 |
Contributors

- Lei Xiong: Leader Developer
- Kang Tian: Developer
- Yuzhe Li: Developer
1.1 Integration

before integration

after SCALEX integration

1.2 Projection

Map new data to the embeddings of reference
A pancreas reference was created by integrating eight batches.
Here, map pancreas_gse81547, pancreas_gse83139 and pancreas_gse114297 to the embeddings of pancreas reference.
1.3 Label transfer

Annotate cells in new data through label transfer

Label transfer tabula muris data and mouse kidney data from mouse atlas reference

mouse atlas reference
query tabula muris aging and query mouse kidney

1.3. Label transfer
1.3. Label transfer
1.4 Integration scATAC-seq data

1.5 Integration cross-modality data

Integrate scRNA-seq and scATAC-seq dataset

1.6 Spatial data (To be updated)

Integrating spatial data with scRNA-seq

1.7 Examples

1.7.1 Integrating PBMC data using SCALEX

The following tutorial demonstrates how to use SCALEX for integrating PBMC data.

There are two parts of this tutorial:

- **Seeing the batch effect.** This part will show the batch effects of two PBMC datasets from single cell 3’ and 5’ gene expression libraries that used in SCALEX manuscript.
• **Integrating data using SCALEX.** This part will show you how to perform batch correction using SCALEX function in SCALEX.

```python
[1]: import scalex
   from scalex.function import SCALEX
   from scalex.plot import embedding
   import scapy as sc
   import pandas as pd
   import numpy as np
   import matplotlib
   from matplotlib import pyplot as plt
   import seaborn as sns
```

```python
[2]: sc.settings.verbosity = 3
   sc.settings.set_figure_params(dpi=80, facecolor='white', figsize=(3,3), frameon=True)
   sc.logging.print_header()
   plt.rcParams['axes.unicode_minus']=False
   scanpy==1.6.1 anndata==0.7.5 umap==0.4.6 numpy==1.20.1 scipy==1.6.1 pandas==1.1.3 scikit-
   →learn==0.23.2 statsmodels==0.12.0 python-igraph==0.8.3 louvain==0.7.0 leidenalg==0.8.3
```

```python
[3]: sns.__version__
[3]: '0.10.1'
```

```python
[4]: scalex.__version__
[4]: '0.2.0'
```

**Seeing the batch effect**

The following data has been used in the Seurat v2 paper, has been used here, and can be downloaded from here. On a unix system, you can uncomment and run the following to download the count matrix in its anndata format.

```python
[5]: # ! wget http://zhanglab.net/scalex-tutorial/pbmc.h5ad
```

```python
[6]: adata_raw=sc.read('pbmc.h5ad')
   adata_raw
```

Anndata object with n_obs × n_vars = 15476 × 33694
obs: 'batch', 'celltype', 'protocol', 'celltype0'

Inspect the batches contained in the dataset.

```python
[7]: adata_raw.obs.batch.value_counts()
```

```python
[7]: pbmc_3p  8098
    pbmc_5p  7378
Name: batch, dtype: int64
```

The data processing procedure is according to the scapy tutorial [Preprocessing and clustering 3k PBMCs].

---

1.7. Examples
[8]:
```python
sc.pp.filter_cells(adata_raw, min_genes=600)
sc.pp.filter_genes(adata_raw, min_cells=3)
adata_raw = adata_raw[:, [gene for gene in adata_raw.var_names if not str(gene).startswith(tuple(['ERCC', 'MT-', 'mt-'])]
sc.pp.normalize_total(adata_raw, target_sum=1e4)
sc.pp.log1p(adata_raw)
sc.pp.highly_variable_genes(adata_raw, min_mean=0.0125, max_mean=3, min_disp=0.5)
adata_raw.raw = adata_raw
adata_raw = adata_raw[:, adata_raw.var.highly_variable]
sc.pp.scale(adata_raw, max_value=10)
sc.pp.pca(adata_raw)
sc.pp.neighbors(adata_raw)
sc.tl.umap(adata_raw)
```

filtered out 13316 genes that are detected in less than 3 cells
normalizing counts per cell
finished (0:00:00)
extracting highly variable genes
finished (0:00:01)
--> added
'highly_variable', boolean vector (adata.var)
'means', float vector (adata.var)
'dispersions', float vector (adata.var)
'dispersions_norm', float vector (adata.var)
... as `zero_center=True`, sparse input is densified and may lead to large memory
consumption
computing PCA
on highly variable genes
with n_comps=50
finished (0:00:02)
computing neighbors
using 'X_pca' with n_pcs = 50
finished: added to `.uns`['neighbors']`
`.obsp`['distances'], distances for each pair of neighbors
`.obsp`['connectivities'], weighted adjacency matrix (0:00:12)
computing UMAP
finished: added
'X_umap', UMAP coordinates (adata.obsm) (0:00:08)

We observe a batch effect.

[9]:
```python
sc.pl.umap(adata_raw,color=['celltype'],legend_fontsize=10)
```
[10]: sc.pl.umap(adata_raw,color=['batch'],legend_fontsize=10)

[11]: adata_raw

AnnData object with n_obs × n_vars = 15476 × 1125
obs: 'batch', 'celltype', 'protocol', 'celltype0', 'n_genes'
    var: 'n_cells', 'highly_variable', 'means', 'dispersions', 'dispersions_norm', 'mean →', 'std'
    uns: 'log1p', 'hvg', 'pca', 'neighbors', 'umap', 'celltype_colors', 'batch_colors'
obsm: 'X_pca', 'X_umap'
varm: 'PCs'
obsp: 'distances', 'connectivities'
Integrating data using SCALEX

The batch effects can be well-resolved using SCALEX.

Note

Here we use GPU to speed up the calculation process, however, you can get the same level of performance only using cpu.

[12]: adata=SCALEX('pbmc.h5ad',batch_name='batch',min_features=600, min_cells=3, outdir='pbmc_output/',show=False,gpu=9)

filtered out 13316 genes that are detected in less than 3 cells

normalizing counts per cell
  finished (0:00:00)

If you pass `n_top_genes`, all cutoffs are ignored.
extracting highly variable genes
  finished (0:00:01)

-> added
  'highly_variable', boolean vector (adata.var)
  'means', float vector (adata.var)
  'dispersions', float vector (adata.var)
  'dispersions_norm', float vector (adata.var)

VAE(
  (encoder): Encoder(
    (enc): NN(
      (net): ModuleList(
        (0): Block(
          (fc): Linear(in_features=2000, out_features=1024, bias=True)
          (norm): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (act): ReLU()
      )
    )
    (mu_enc): NN(
      (net): ModuleList(
        (0): Block(
          (fc): Linear(in_features=1024, out_features=10, bias=True)
        )
      )
    )
  )
(continues on next page)
(var_enc): NN(
  (net): ModuleList(
    (0): Block(
      (fc): Linear(in_features=1024, out_features=10, bias=True)
    )
  )
)

(decoder): NN(
  (net): ModuleList(
    (0): Block(
      (fc): Linear(in_features=10, out_features=2000, bias=True)
      (norm): DSBatchNorm(
        (bns): ModuleList(
          (0): BatchNorm1d(2000, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (1): BatchNorm1d(2000, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (act): Sigmoid()
    )
  )
)

Epochs: 100% | 125/125 [06:11<00:00, 2.97s/it, recon_loss=182.388, kl_loss=3.988]
2021-03-30 20:23:03,050 - root - INFO - Output dir: pbmc_output/
2021-03-30 20:23:07,564 - root - INFO - Plot umap

computing neighbors
  finished: added to `.uns['neighbors']`
  `.obsp['distances']`, distances for each pair of neighbors
  `.obsp['connectivities']`, weighted adjacency matrix (0:00:03)
computing UMAP
  finished: added
  'X_umap', UMAP coordinates (adata.obsm) (0:00:12)
running Leiden clustering
  finished: found 15 clusters and added
  'leiden', the cluster labels (adata.obs, categorical) (0:00:05)
WARNING: saving figure to file pbmc_output/umap.pdf

[13]: adata

[13]: AnnData object with n_obs × n_vars = 15476 × 2000
  obs: 'batch', 'celltype', 'protocol', 'celltype0', 'n_genes', 'leiden'
  var: 'n_cells', 'highly_variable', 'means', 'dispersions', 'dispersions_norm',
  'highly_variable_nbatches', 'highly_variable_intersection'
  uns: 'log1p', 'hvg', 'neighbors', 'umap', 'leiden', 'batch_colors', 'celltype_colors'
  obsm: 'latent', 'X_umap'

(continues on next page)
While there seems to be some strong batch-effect in all cell types, SCALEX can integrate them homogeneously.

```python
[14]: sc.settings.set_figure_params(dpi=80, facecolor='white', figsize=(3,3), frameon=True)
```

```python
[15]: sc.pl.umap(adata,color=['celltype'],legend_fontsize=10)
```

![Celltype UMAP](image)

```python
[16]: sc.pl.umap(adata,color=['batch'],legend_fontsize=10)
```

![Batch UMAP](image)

The integrated data is stored as `adata.h5ad` in the output directory assigned by `outdir` parameter in `SCALEX` function.
1.7.2 Projection pancreas data using SCALEX

The following tutorial demonstrates how to use SCALEX for integrating data and projection new data adata_query onto an annotated reference adata_ref.

There are five parts of this tutorial:

• **Seeing the batch effect.** This part will show the batch effects of eight pancreas datasets that used in SCALEX manuscript.

• **Integrating data using SCALEX.** This part will show you how to perform batch correction and construct a reference batch adata_ref using SCALEX function in SCALEX.

• **Mapping onto a reference batch using projection function.** The third part will describe the usage of projection function in SCALEX to map three batches query dataset adata_query onto the reference batch adata_ref you construted in part two.

• **Visualizing distributions across batches.** Often, batches correspond to experiments that one wants to compare. SCALEX v2 offers embedding function to convenient visualize for this.

• **Label transfer.** SCALEX offers label_transfer function to conveniently transfer labels from reference datasets to query datasets.

```python
[1]: import scalex
    from scalex import SCALEX, label_transfer
    from scalex.plot import embedding
    import scanpy as sc
    import numpy as np
    import matplotlib
    from matplotlib import pyplot as plt
    import seaborn as sns

[2]: sc.settings.verbosity = 3
    sc.settings.set_figure_params(dpi=80, facecolor='white', figsize=(3,3), frameon=True)
    sc.logging.print_header()
    plt.rcParams['axes.unicode_minus']=False
    scanpy==1.6.1 anndata==0.7.5 umap==0.4.6 numpy==1.20.1 scipy==1.6.1 pandas==1.1.3 scikit-learn==0.23.2 statsmodels==0.12.0 python-igraph==0.8.3 louvain==0.7.0 leidenalg==0.8.3

[3]: sns.__version__
[3]: '0.10.1'

[4]: scalex.__version__
[4]: '0.2.0'
```
**Seeing the batch effect**

The pancreas data has been used in the Seurat v3 and Harmony paper.

On a unix system, you can uncomment and run the following to download the count matrix in its anndata format.

```bash
[5]: # ! wget http://zhanglab.net/scalex-tutorial/pancreas.h5ad
    # ! wget http://zhanglab.net/scalex-tutorial/pancreas_query.h5ad
```

```python
[6]: adata_raw=sc.read('pancreas.h5ad')
    adata_raw
```

```
AnnData object with n_obs × n_vars = 16401 × 14895
    obs: 'batch', 'celltype', 'disease', 'donor', 'library', 'protocol'
```

Inspect the batches contained in the dataset.

```python
[7]: adata_raw.obs.batch.value_counts()
```

```
pancreas_indrop3   3605
pancreas_celseq2   3072
pancreas_smartseq2 2394
pancreas_indrop1   1937
pancreas_celseq    1728
pancreas_indrop2   1724
pancreas_indrop4   1303
pancreas_fluidigmc1 638
Name: batch, dtype: int64
```

The data processing procedure is according to the scanpy tutorial [Preprocessing and clustering 3k PBMCs].

```python
[8]: sc.pp.filter_cells(adata_raw, min_genes=600)
    sc.pp.filter_genes(adata_raw, min_cells=3)
    adata_raw = adata_raw[:, [gene for gene in adata_raw.var_names if not str(gene).startswith(tuple(['ERCC', 'MT-', 'mt-']))]]
    sc.pp.normalize_total(adata_raw, target_sum=1e4)
    sc.pp.log1p(adata_raw)
    sc.pp.highly_variable_genes(adata_raw, min_mean=0.0125, max_mean=3, min_disp=0.5)
    adata_raw.raw = adata_raw
    adata_raw = adata_raw[:, adata_raw.var.highly_variable]
    sc.pp.scale(adata_raw, max_value=10)
    sc.pp.pca(adata_raw)
    sc.pp.neighbors(adata_raw)
    sc.tl.umap(adata_raw)
```

filtered out 1124 cells that have less than 600 genes expressed
normalizing counts per cell
    finished (0:00:00)
extracting highly variable genes
    finished (0:00:01)
    added
        'highly_variable', boolean vector (adata.var)
        'means', float vector (adata.var)
        'dispersions', float vector (adata.var)
        'dispersions_norm', float vector (adata.var)
```

(continues on next page)
... as `zero_center=True`, sparse input is densified and may lead to large memory consumption.

computing PCA
   on highly variable genes
   with n_comps=50
   finished (0:00:03)

computing neighbors
   using 'X_pca' with n_pcs = 50
   finished: added to `.uns['neighbors']`
   `.obsp['distances']`, distances for each pair of neighbors
   `.obsp['connectivities']`, weighted adjacency matrix (0:00:15)

computing UMAP
   finished: added
   'X_umap', UMAP coordinates (adata.obsm) (0:00:09)

We observe a batch effect.

[9]: sc.pl.umap(adata_raw,color=['celltype'],legend_fontsize=10)

![celltype UMAP](image1)

[10]: sc.pl.umap(adata_raw,color=['batch'],legend_fontsize=10)

![batch UMAP](image2)
Integrating data using SCALEX

The batch effects can be well-resolved using SCALEX.

Note

Here we use GPU to speed up the calculation process, however, you can get the same level of performance only using cpu.

If you pass `n_top_genes`, all cutoffs are ignored.

extracting highly variable genes

finished (0:00:04)

--> added

'highly_variable', boolean vector (adata.var)
'means', float vector (adata.var)
'dispersions', float vector (adata.var)
'dispersions_norm', float vector (adata.var)
(net): ModuleList(
    (0): Block(
        (fc): Linear(in_features=2000, out_features=1024, bias=True)
        (norm): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (act): ReLU()
    )
)

(mu_enc): NN(
    (net): ModuleList(
        (0): Block(
            (fc): Linear(in_features=1024, out_features=10, bias=True)
        )
    )
)

(var_enc): NN(
    (net): ModuleList(
        (0): Block(
            (fc): Linear(in_features=1024, out_features=10, bias=True)
        )
    )
)

(decoder): NN(
    (net): ModuleList(
        (0): Block(
            (fc): Linear(in_features=10, out_features=2000, bias=True)
            (norm): DSBatchNorm(
                (bns): ModuleList(
                    (0): BatchNorm1d(2000, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
                    (1): BatchNorm1d(2000, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
                    (2): BatchNorm1d(2000, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
                    (3): BatchNorm1d(2000, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
                    (4): BatchNorm1d(2000, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
                    (5): BatchNorm1d(2000, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
                    (6): BatchNorm1d(2000, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
                    (7): BatchNorm1d(2000, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
                )
            )
            (act): Sigmoid()
        )
    )
)
Epochs: 100% | 127/127 [09:11<00:00, 4.34s/it, recon_loss=266.572, kl_loss=4.667]
2021-03-30 20:30:38,361 - root - INFO - Output dir: pancreas_output//
2021-03-30 20:30:47,018 - root - INFO - Plot umap computing neighbors
  finished: added to `.uns['neighbors']`
  `.obsp['distances']`, distances for each pair of neighbors
  `.obsp['connectivities']`, weighted adjacency matrix (0:00:03)
computing UMAP
  finished: added
 'X_umap', UMAP coordinates (adata.obsm) (0:00:10)
running Leiden clustering
  finished: found 13 clusters and added
 'leiden', the cluster labels (adata.obs, categorical) (0:00:02)
WARNING: saving figure to file pancreas_output/umap.pdf

While there seems to be some strong batch-effect in all cell types, SCALEX can integrate them homogeneously.

```python
[13]: adata_ref
AnnData object with n_obs × n_vars = 15277 × 2000
  obs: 'batch', 'celltype', 'disease', 'donor', 'library', 'protocol', 'n_genes', 'leiden'
  var: 'n_cells', 'highly_variable', 'means', 'dispersions', 'dispersions_norm', 'highly_variable_nbatches', 'highly_variable_intersection'
  uns: 'log1p', 'hvg', 'neighbors', 'umap', 'leiden', 'batch_colors', 'celltype_colors', 'leiden_colors'
  obsm: 'latent', 'X_umap'
  obsp: 'distances', 'connectivities'
```

```python
[14]: sc.settings.set_figure_params(dpi=80, facecolor='white', figsize=(3,3), frameon=True)
[15]: sc.pl.umap(adata_ref,color=['celltype'],legend_fontsize=10)
```

![UMAP Plot](image)

```python
[16]: sc.pl.umap(adata_ref,color=['batch'],legend_fontsize=10)
```
The integrated data is stored as `adata.h5ad` in the output directory assigned by `outdir` parameter in `SCALEX` function.

### Mapping onto a reference batch using projection function

The pancreas query data are available through the Gene Expression Omnibus under accession GSE114297, GSE81547 and GSE83139.

```bash
[17]: adata_query=sc.read('pancreas_query.h5ad')
adata_query
```

```bash
AnnData object with n_obs × n_vars = 23963 × 31884
obs: 'batch', 'celltype', 'disease', 'donor', 'protocol'
```

Inspect the batches contained in `adata_query`.

```bash
[18]: adata_query.obs.batch.value_counts()
```

```bash
pancreas_gse114297    20784
pancreas_gse81547     2544
pancreas_gse83139     635
Name: batch, dtype: int64
```

`SCALEX` provides a projection function for mapping new data `adata_query` onto the reference batch `adata_ref`.

```bash
[19]: adata=SCALEX('pancreas_query.h5ad',batch_name='batch',min_features=600,min_cells=3,
                  outdir='pancreas_projection/',projection='pancreas_output/',show=False,
                  →gpu=7)
```

```bash
2021-03-30 20:31:47,177 - root - INFO - Raw dataset shape: (23963, 31884)
2021-03-30 20:31:47,177 - root - INFO - Raw dataset shape: (23963, 31884)
2021-03-30 20:31:47,180 - root - INFO - Preprocessing
2021-03-30 20:31:47,180 - root - INFO - Preprocessing
2021-03-30 20:31:47,236 - root - INFO - Filtering cells
2021-03-30 20:31:47,236 - root - INFO - Filtering cells
filtered out 219 cells that have less than 600 genes expressed
```
Setting attribute `.obs` of view, copying.
2021-03-30 20:31:49,744 - root - INFO - Filtering features
2021-03-30 20:31:49,744 - root - INFO - Filtering features
2021-03-30 20:31:51,094 - root - INFO - Normalizing total per cell
2021-03-30 20:31:51,094 - root - INFO - Normalizing total per cell

Filtering features
Normalizing total per cell
finished (0:00:00)

Log1p transforming
Finding variable features
Finding variable features

There are 2000 gene in selected genes
Batch specific maxabs scaling
Processed dataset shape: (23744, 2000)
Processed dataset shape: (23744, 2000)
Output dir: pancreas_projection/
Output dir: pancreas_projection/

... storing 'batch' as categorical
... storing 'celltype' as categorical
... storing 'disease' as categorical
... storing 'donor' as categorical
... storing 'library' as categorical
... storing 'protocol' as categorical
... storing 'leiden' as categorical

computing neighbors
finished: added to `.uns['neighbors']`
`.obsp['distances']`, distances for each pair of neighbors
`.obsp['connectivities']`, weighted adjacency matrix (0:00:08)
computing UMAP
finished: added
'X_umap', UMAP coordinates (adata.obsm) (0:00:26)
running Leiden clustering
finished: found 13 clusters and added
'leiden', the cluster labels (adata.obs, categorical) (0:00:10)
WARNING: saving figure to file pancreas_projection/X_umap_reference.pdf
WARNING: saving figure to file pancreas_projection/X_umap_query.pdf

Load integrated data `adata` that contained `adata_ref` and `adata_query`.

```python
[20]: sc.settings.set_figure_params(dpi=80, facecolor='white',figsize=(3,3),frameon=True)
[21]: embedding(adata, groupby='projection')
```
Inspect the batches contained in `adata`.

```python
adata.obs.batch.value_counts()
```

```
pancreas_gse114297 20573
pancreas_indrop3 3605
pancreas_gse81547 2536
pancreas_celseq2 2440
pancreas_smartseq2 2394
pancreas_indrop1 1937
```

(continues on next page)
pancreas_indrop2 1724
pancreas_indrop4 1303
pancreas_celseq 1236
pancreas_fluidigmcl 638
pancreas_gse83139 635
Name: batch, dtype: int64

Visualizing distributions across batches

```python
[24]: sc.pl.umap(adata[adata.obs.batch.isin(['pancreas_gse114297','pancreas_gse81547','pancreas_gse83139'])],color='batch',legend_fontsize=10)
embedding(adata[adata.obs.batch.isin(['pancreas_gse114297','pancreas_gse81547','pancreas_gse83139'])],legend_fontsize=10)
```

Trying to set attribute `.uns` of view, copying.

![UMAP 1 and UMAP 2 plots showing distribution across batches](image)

Trying to set attribute `.obs` of view, copying.

![X_umap1 and X_umap2 plots showing cell types](image)
The projection results is stored as `adata.h5ad` in the output directory assigned by `outdir` parameter in `SCALE` function.

**Label transfer**

We can also use `SCALEX` to transfer data from one dataset to another. Here, we demonstrate data transfer between two scRNA-seq datasets by transferring the cell type label from the `adata_ref` and the `adata_query`.

```bash
[25]: adata_query=adata[adata.obs.projection=="query"]
adata_query
```

```bash
[25]: View of AnnData object with n_obs × n_vars = 23744 × 2000
    obs: 'batch', 'celltype', 'disease', 'donor', 'library', 'protocol', 'n_genes', 'leiden', 'projection'
    uns: 'neighbors', 'umap', 'leiden'
    obsm: 'latent', 'X_umap'
    obsp: 'distances', 'connectivities'
```
[26]: adata_ref = adata[adata.obs.projection=='reference']
adata_ref

[26]: View of Anndata object with n_obs x n_vars = 15277 x 2000
obs: 'batch', 'celltype', 'disease', 'donor', 'library', 'protocol', 'n_genes',
    'leiden', 'projection'
var: 'n_cells-reference', 'highly_variable-reference', 'means-reference',
    'dispersions-reference', 'dispersions_norm-reference', 'highly_variable_nbatches-
    reference', 'highly_variable_intersection-reference'
uns: 'neighbors', 'umap', 'leiden'
obsm: 'latent', 'X_umap'
obsp: 'distances', 'connectivities'

[27]: adata_query.obs['celltype_transfer']=label_transfer(adata_ref, adata_query, rep='latent',
                                          label='celltype')

Trying to set attribute `.obs` of view, copying.

[28]: sc.pl.umap(adata_query,color=['celltype_transfer'])

... storing 'celltype_transfer' as categorical

[29]: sc.pl.umap(adata_query,color=['celltype'])
Let us first focus on cell types that are conserved with the reference.

```python
obs_query = adata_query.obs
conserved_categories = obs_query.celltype.cat.categories.intersection(obs_query.celltype_transfer.cat.categories) # intersected categories
obs_query_conserved = obs_query.loc[obs_query.celltype.isin(conserved_categories) & obs_query.celltype_transfer.isin(conserved_categories)] # intersect categories
obs_query_conserved.celltype.cat.remove_unused_categories(inplace=True) # remove unused categories
obs_query_conserved.celltype_transfer.cat.remove_unused_categories(inplace=True) # remove unused categories
obs_query_conserved.celltype_transfer.cat.reorder_categories(obs_query_conserved.celltype.cat.categories, inplace=True) # fix category ordering
pd.crosstab(obs_query_conserved.celltype, obs_query_conserved.celltype_transfer)
```

<table>
<thead>
<tr>
<th>celltype_transfer</th>
<th>acinar</th>
<th>activated_stellate</th>
<th>alpha</th>
<th>alpha_er</th>
<th>beta</th>
</tr>
</thead>
<tbody>
<tr>
<td>acinar</td>
<td>1346</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>activated_stellate</td>
<td>0</td>
<td>1033</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>alpha</td>
<td>2</td>
<td>0</td>
<td>7527</td>
<td>83</td>
<td>13</td>
</tr>
<tr>
<td>alpha_er</td>
<td>0</td>
<td>0</td>
<td>159</td>
<td>105</td>
<td>2</td>
</tr>
<tr>
<td>beta</td>
<td>3</td>
<td>0</td>
<td>119</td>
<td>4</td>
<td>8683</td>
</tr>
<tr>
<td>beta_er</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>41</td>
</tr>
<tr>
<td>delta</td>
<td>1</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>34</td>
</tr>
<tr>
<td>ductal</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>endothelial</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>epsilon</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>gamma</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>macrophage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mast</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>quiescent_stellate</td>
<td>0</td>
<td>53</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>schwann</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>celltype_transfer</th>
<th>beta_er</th>
<th>delta</th>
<th>ductal</th>
<th>endothelial</th>
<th>epsilon</th>
<th>gamma</th>
</tr>
</thead>
<tbody>
<tr>
<td>acinar</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>activated_stellate</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>alpha</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>alpha_er</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>beta</td>
<td>34</td>
<td>17</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>beta_er</td>
<td>56</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>delta</td>
<td>0</td>
<td>1099</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>ductal</td>
<td>0</td>
<td>3</td>
<td>1889</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>endothelial</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>456</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>epsilon</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>gamma</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>578</td>
</tr>
<tr>
<td>macrophage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mast</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>quiescent_stellate</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>schwann</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>celltype_transfer</th>
<th>macrophage</th>
<th>mast</th>
<th>quiescent_stellate</th>
<th>schwann</th>
</tr>
</thead>
<tbody>
<tr>
<td>acinar</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

(continues on next page)
Let us now move on to look at all cell types.

```python
[31]: pd.crosstab(adata_query.obs.celltype, adata_query.obs.celltype_transfer)

<table>
<thead>
<tr>
<th>celltype_transfer</th>
<th>acinar</th>
<th>activated_stellate</th>
<th>alpha</th>
<th>alpha_er</th>
<th>beta</th>
<th>beta_er</th>
<th>delta</th>
<th>ductal</th>
<th>endothelial</th>
<th>epsilon</th>
<th>gamma</th>
<th>macrophage</th>
<th>mast</th>
<th>quiescent_stellate</th>
<th>schwann</th>
</tr>
</thead>
<tbody>
<tr>
<td>acinar</td>
<td></td>
<td>1346</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>activated_stellate</td>
<td>0</td>
<td>1033</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>alpha</td>
<td>2</td>
<td>0</td>
<td>7527</td>
<td>83</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>8683</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>alpha_er</td>
<td>0</td>
<td>0</td>
<td>159</td>
<td>105</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>beta</td>
<td>3</td>
<td>0</td>
<td>119</td>
<td>4</td>
<td>8683</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>beta_er</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>41</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delta</td>
<td>1</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>34</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ductal</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>endothelial</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>epsilon</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gamma</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>macrophage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mast</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>quiescent_stellate</td>
<td>0</td>
<td>53</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>schwann</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```python
[31]: celltype_transfer beta_er delta ductal endothelial epithelial epsilon \celltype
| acinar             | 0     | 0     | 2       | 0     | 0     | 0       |          |
| activated_stellate | 0     | 0     | 0       | 0     | 0     | 0       |          |
| alpha              | 0     | 3     | 0       | 0     | 1     | 0       |          |
| alpha_er           | 1     | 0     | 0       | 0     | 0     | 0       |          |
| beta               | 34    | 17    | 1       | 0     | 0     | 0       |          |
| beta_er            | 56    | 0     | 1       | 0     | 0     | 0       |          |
| delta              | 0     | 1099  | 0       | 0     | 0     | 1       |          |
| ductal             | 0     | 3     | 1889    | 0     | 0     | 0       |          |
| endothelial        | 0     | 1     | 0       | 456   | 0     | 0       |          |
| epsilon            | 0     | 3     | 0       | 0     | 10    | 0       |          |
| gamma              | 0     | 1     | 0       | 0     | 0     | 2       |          |
| macrophage         | 0     | 0     | 0       | 0     | 0     | 0       |          |
| mast               | 0     | 0     | 0       | 1     | 0     | 0       |          |
| quiescent_stellate | 0     | 0     | 0       | 0     | 0     | 0       |          |
```

(continues on next page)
1.7.3 Integrating scATAC-seq data using SCALEX

The following tutorial demonstrates how to use SCALEX for integrating scATAC-seq data.

There are two parts of this tutorial:

- **Seeing the batch effect.** This part will show the batch effects of two adult mouse brain datasets from single nucleus ATAC-seq (snATAC) and droplet-based platform (Mouse Brain 10X) that used in SCALEX manuscript.

- **Integrating data using SCALEX.** This part will show you how to perform batch correction using SCALEX function in SCALEX.

[1]: import scalex
from scalex.function import SCALEX
from scalex.plot import embedding
import scanpy as sc
import pandas as pd
import numpy as np
import matplotlib
from matplotlib import pyplot as plt
import seaborn as sns
import episcanpy as epi

[2]: sc.settings.verbosity = 3
sc.settings.set_figure_params(dpi=80, facecolor='white',figsize=(3,3),frameon=True)
sc.logging.print_header()
plt.rcParams['axes.unicode_minus']=False
scanpy==1.6.1 anndata==0.7.5 umap==0.4.6 numpy==1.20.1 scipy==1.6.1 pandas==1.1.3 scikit_...learn==0.23.2 statsmodels==0.12.0 python-igraph==0.8.3 louvain==0.7.0 leidenalg==0.8.3

[3]: sns.__version__
Seeing the batch effect

The following data has been used in the SnapATAC paper, has been used here, and can be downloaded from here. On a unix system, you can uncomment and run the following to download the count matrix in its anndata format.

```python
# ! wget http://zhanglab.net/scalex-tutorial/mouse_brain_atac.h5ad
```

```python
adata_raw=sc.read('mouse_brain_atac.h5ad')
adata_raw
```

AnnData object with n_obs × n_vars = 13746 × 479127
obs: 'batch'

Inspect the batches contained in the dataset.

```python
adata_raw.obs.batch.value_counts()
```

snATAC 9646 10X 4100
Name: batch, dtype: int64

The data processing procedure is according to the epiScanpy tutorial [Buenrostro_PBMC_data_processing].

```python
epi.pp.filter_cells(adata_raw, min_features=1)
epi.pp.filter_features(adata_raw, min_cells=5)
adatraw.raw = adata_raw
adata_raw = epi.pp.select_var_feature(adata_raw, nb_features=30000, show=False, copy=True)
adatraw.layers['binary'] = adata_raw.X.copy()
epi.pp.normalize_total(adata_raw)
adatraw.layers['normalised'] = adata_raw.X.copy()
epi.pp.log1p(adata_raw)
epi.pp.lazy(adata_raw)
epi.tl.leiden(adata_raw)
```

filtered out 11140 genes that are detected in less than 5 cells
normalizing counts per cell
finished (0:00:00)
computing PCA
with n_comps=50
finished (0:01:46)
computing neighbors
using 'X_pca' with n_pcs = 50
finished: added to `.uns['neighbors']`
`.obsp['distances']`, distances for each pair of neighbors
`.obsp['connectivities']`, weighted adjacency matrix (0:00:14)

(continues on next page)
computing tSNE
    using 'X_pca' with n_pcs = 50
    using the 'MulticoreTSNE' package by Ulyanov (2017)
    finished: added
    'X_tsne', tSNE coordinates (adata.obsm) (0:01:35)
computing UMAP
    finished: added
    'X_umap', UMAP coordinates (adata.obsm) (0:00:09)
running Leiden clustering
    finished: found 25 clusters and added
    'leiden', the cluster labels (adata.obs, categorical) (0:00:01)

We observe a batch effect.

[9]: sc.pl.umap(adata_raw,color=['leiden'],legend_fontsize=10)

---

[10]: sc.pl.umap(adata_raw,color=['batch'],legend_fontsize=10)

---

[11]: adata_raw
[11]: AnnData object with n_obs x n_vars = 13746 x 30076
Integrating data using SCALEX

The batch effects can be well-resolved using SCALEX.

**Note**

Here we use GPU to speed up the calculation process, however, you can get the same level of performance only using cpu.

```python
[12]: adata=SCALEX('mouse_brain_atac.h5ad',batch_name='batch',profile='ATAC',
min_features=1, min_cells=5, n_top_features=30000,outdir='ATAC_output/',
   show=False,gpu=8)
```

```python
2021-03-30 20:21:45,161 - root - INFO - Raw dataset shape: (13746, 479127)
2021-03-30 20:21:45,166 - root - INFO - Preprocessing
2021-03-30 20:21:45,639 - root - INFO - Filtering cells
2021-03-30 20:21:47,224 - root - INFO - Filtering features
filtered out 11140 genes that are detected in less than 5 cells
2021-03-30 20:21:49,461 - root - INFO - Finding variable features
2021-03-30 20:21:51,582 - root - INFO - Normalizing total per cell
normalizing counts per cell
finished (0:00:00)
2021-03-30 20:21:51,690 - root - INFO - Batch specific maxabs scaling
2021-03-30 20:22:16,926 - root - INFO - Processed dataset shape: (13746, 30076)
2021-03-30 20:22:17,234 - root - INFO - model
VAE(
  (encoder): Encoder(
    (enc): NN(
      (net): ModuleList(
        (0): Block(
          (fc): Linear(in_features=30076, out_features=1024, bias=True)
          (norm): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (act): ReLU()
        )
      )
    )
  )
  (mu_enc): NN(
    (net): ModuleList(
      (0): Block(
```
(fc): Linear(in_features=1024, out_features=10, bias=True)

(var_enc): NN(
  (net): ModuleList(
    (0): Block(
      (fc): Linear(in_features=1024, out_features=10, bias=True)
    )
  )
)

(decoder): NN(
  (net): ModuleList(
    (0): Block(
      (fc): Linear(in_features=10, out_features=30076, bias=True)
      (norm): DSBatchNorm(
        (bns): ModuleList(
          (0): BatchNorm1d(30076, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (1): BatchNorm1d(30076, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (act): Sigmoid()
    )
  )
)

Epochs: 100% | 141/141 [10:53<00:00, 4.63s/it, recon_loss=992.783, kl_loss=3.643]
2021-03-30 20:33:17,866 - root - INFO - Output dir: ATAC_output/
2021-03-30 20:33:22,489 - root - INFO - Plot umap

computing neighbors
  finished: added to `.uns['neighbors']`
  `.obsp['distances']`, distances for each pair of neighbors
  `.obsp['connectivities']`, weighted adjacency matrix (0:00:03)
computing UMAP
  finished: added
  'X_umap', UMAP coordinates (adata.obsm) (0:00:09)
running Leiden clustering
  finished: found 16 clusters and added
  'leiden', the cluster labels (adata.obs, categorical) (0:00:01)
WARNING: saving figure to file ATAC_output/umap.pdf

[13]: adata

<table>
<thead>
<tr>
<th>13:</th>
<th>AnnData object with n_obs × n_vars = 13746 × 30076</th>
</tr>
</thead>
<tbody>
<tr>
<td>obs:</td>
<td>'batch', 'n_genes', 'leiden'</td>
</tr>
<tr>
<td>var:</td>
<td>'n_cells', 'prop_shared_cells', 'variability_score'</td>
</tr>
<tr>
<td>uns:</td>
<td>'neighbors', 'umap', 'leiden', 'batch_colors', 'leiden_colors'</td>
</tr>
<tr>
<td>obsm:</td>
<td>'latent', 'X_umap'</td>
</tr>
<tr>
<td>obsp:</td>
<td>'distances', 'connectivities'</td>
</tr>
</tbody>
</table>
While there seems to be some strong batch-effect in all cell types, SCALEX can integrate them homogeneously.

```python
[14]: sc.settings.set_figure_params(dpi=80, facecolor='white', figsize=(3,3), frameon=True)
```

```python
[15]: sc.pl.umap(adata,color=['leiden'],legend_fontsize=10)
```

![Leiden UMAP plot](image)

```python
[16]: sc.pl.umap(adata,color=['batch'],legend_fontsize=10)
```

![Batch UMAP plot](image)

The integrated data is stored as `adata.h5ad` in the output directory assigned by `outdir` parameter in `SCALE` function.

### 1.7.4 Integration cross-modality data using SCALEX

The following tutorial demonstrates how to use SCALEX for integrating scRNA-seq and scATAC-seq data.

There are mainly two steps:

- **Create a gene activity matrix from scATAC-seq data.** This step follows the standard workflow of Signac for scATAC-seq data analysis. We use the function `GeneActivity` of Signac and calculate the activity of each gene in the genome by assessing the chromatin accessibility associated with each gene, and create a new gene activity matrix derived from the scATAC-seq data. More details are here.

- **Integrate.** We regard gene expression matrix and gene activity matrix as two batches of one dataset and use SCALEX for integration.
For this tutorial, we used a cross-modality PBMC data between scRNA-seq and scATAC-seq provided by 10X Genomics, and both scRNA-seq and scATAC-seq data are available through the 10x Genomics website.

**Create a gene activity matrix (R)**

```
[1]: suppressPackageStartupMessages(library(Signac))
suppressPackageStartupMessages(library(Seurat))
suppressPackageStartupMessages(library(GenomeInfoDb))
suppressPackageStartupMessages(library(EnsDb.Hsapiens.v75))
suppressPackageStartupMessages(library(ggplot2))
suppressPackageStartupMessages(library(patchwork))
set.seed(1234)
options(warn=-1)
```

**Pre-processing**

We follow the pre-processing workflow of Signac when pre-processing chromatin data. First we creat a Seurat object by two related input files: cell matrix and fragment file.

```
[2]: counts <- Read10X_h5(filename = "atac_v1_pbmc_10k_filtered_peak_bc_matrix.h5")
metadata <- read.csv(file = "atac_v1_pbmc_10k_singlecell.csv", header = TRUE, row.names = 1)

chrom_assay <- CreateChromatinAssay(
counts = counts,
sep = c(":", ":"),
genome = "hg19",
fragments = "atac_v1_pbmc_10k.fragments.tsv.gz",
min.cells = 10,
min.features = 200
)

pbmc <- CreateSeuratObject(
counts = chrom_assay,
assay = "peaks",
meta.data = metadata
)
```

**Computing hash**

Now add gene annotations to the pbmc object for the human genome. genome is suggested to be as same as the genome for scRNA-seq reads mapping on.

```
[4]: # extract gene annotations from EnsDb
annotations <- GetGRangesFromEnsDb(ensdb = EnsDb.Hsapiens.v75)

# change to UCSC style since the data was mapped to hg19
seqlevelsStyle(annotations) <- 'UCSC'
```
genome(annotations) <- "hg19"

# add the gene information to the object
Annotation(pbmc) <- annotations

Compute QC Metrics. If you don’t need to filter cells, ignore this step

[13]: # compute nucleosome signal score per cell
pbmc <- NucleosomeSignal(object = pbmc)

# compute TSS enrichment score per cell
pbmc <- TSSEnrichment(object = pbmc, fast = FALSE)

# add blacklist ratio and fraction of reads in peaks
pbmc$pct_reads_in_peaks <- pbmc$peak_region_fragments / pbmc$passed_filters * 100
pbmc$blacklist_ratio <- pbmc$blacklist_region_fragments / pbmc$peak_region_fragments

Extracting TSS positions
Finding + strand cut sites
Finding - strand cut sites
Computing mean insertion frequency in flanking regions
Normalizing TSS score

[14]: pbmc$high.tss <- ifelse(pbmc$TSS.enrichment > 2, 'High', 'Low')
TSSPlot(pbmc, group.by = 'high.tss') + NoLegend()
[15]: pbmc$nucleosome_group <- ifelse(pbmc$nucleosome_signal > 4, 'NS > 4', 'NS < 4')
FragmentHistogram(object = pbmc, group.by = 'nucleosome_group')
[16]: VlnPlot(
    object = pbmc,
    features = c('pct_reads_in_peaks', 'peak_region_fragments',
                  'TSS.enrichment', 'blacklist_ratio', 'nucleosome_signal'),
    pt.size = 0.1,
    ncol = 5
)
Filter cells that are outliers for the QC metrics.

```
[17]: pbmc <- subset(
  x = pbmc,
  subset = peak_region_fragments > 3000 &
          peak_region_fragments < 20000 &
          pct_reads_in_peaks > 15 &
          blacklist_ratio < 0.05 &
          nucleosome_signal < 4 &
          TSS.enrichment > 2
)
```

An object of class Seurat
87561 features across 7060 samples within 1 assay
Active assay: peaks (87561 features, 0 variable features)
Calculate gene activity matrix by `GeneActivity()` function

```r
[5]: gene.activities <- GeneActivity(pbmc)
```

Extracting gene coordinates

Extracting reads overlapping genomic regions

```r
save results

[ ]: wk_dir <- './'
```

```r
[2]: write.table(t(gene.activities), paste(wk_dir,"gene_activity_score.txt",sep=''), sep='\t', quote = F)
```

Integrate (python)

Now you can use the `gene_activity_score` together with the gene expression matrix for integration.

You can run SCALE command line directly: SCALEX.py --data_list data1 data2 --batch_categories RNA ATAC -o output_path

- **data1**: path or file of scRNA-seq data
- **data2**: file of `gene_activity_score`

and here we show the results before integration and after integration.

```python
[3]: import scalex
from scalex import SCALEX
from scalex.plot import embedding
import scanpy as sc
import numpy as np
import matplotlib
from matplotlib import pyplot as plt
import seaborn as sns
```

```python
[19]: sc.settings.verbosity = 3
sc.settings.set_figure_params(dpi=80, facecolor='white', figsize=(3,3), frameon=True)
sc.logging.print_header()
plt.rcParams['axes.unicode_minus'] = False
```

```python
[3]: scalex.__version__
```

'2.0.3.dev'

First we merge the RNA and ATAC data and add metadata information, and this processed data is available here

```r
[ ]: wk_dir = './'
```
[10]: adata = sc.read_h5ad(wk_dir+'pbmc_RNA-ATAC.h5ad)

[11]: adata

AnnData object with n_obs × n_vars = 16492 × 13928
   obs: 'celltype', 'tech', 'batch'

[12]: sc.pp.filter_cells(adata, min_genes=0)
    sc.pp.filter_genes(adata, min_cells=0)
    sc.pp.normalize_total(adata, inplace=True)
    sc.pp.log1p(adata)
    sc.pp.highly_variable_genes(adata, n_top_genes=2000)
    adata = adata[:, adata.var.highly_variable]
    sc.pp.scale(adata, max_value=10)
    sc.tl.pca(adata)
    sc.pp.neighbors(adata, n_pcs=30, n_neighbors=30)
    sc.tl.umap(adata, min_dist=0.1)

normalizing counts per cell
   finished (0:00:00)
If you pass `n_top_genes`, all cutoffs are ignored.
extracting highly variable genes
   finished (0:00:01)
   --> added
       'highly_variable', boolean vector (adata.var)
       'means', float vector (adata.var)
       'dispersions', float vector (adata.var)
       'dispersions_norm', float vector (adata.var)
... as `zero_center=True`, sparse input is densified and may lead to large memory consumption
computing PCA
   on highly variable genes
   with n_comps=50
   finished (0:00:02)
computing neighbors
   using 'X_pca' with n_pcs = 30
   finished: added to `.uns['neighbors']`
   `.obsp['distances']`, distances for each pair of neighbors
   `.obsp['connectivities']`, weighted adjacency matrix (0:00:14)
computing UMAP
   finished: added
   'X_umap', UMAP coordinates (adata.obsm) (0:00:29)

[15]: sc.pl.umap(adata,color=['batch','celltype'],legend_fontsize=10, ncols=2)
[16]: wk_dir='/' # wk_dir is your local path to store data and results

[17]: adata = SCALEX(data_list = [wk_dir+'RNA-ATAC.h5ad'],
    min_features=0,
    min_cells=0,
    outdir=wk_dir+'/pbmc_RNA_ATAC/',
    show=False,
    gpu=7)

2021-03-26 11:46:17,234 - root - INFO - Raw dataset shape: (16492, 13928)
2021-03-26 11:46:17,237 - root - INFO - Preprocessing
2021-03-26 11:46:21,627 - root - INFO - Filtering cells
Trying to set attribute `_.obs` of view, copying.
2021-03-26 11:46:24,745 - root - INFO - Normalizing total per cell
normalizing counts per cell
    finished (0:00:00)
2021-03-26 11:46:25,022 - root - INFO - Log1p transforming
2021-03-26 11:46:26,014 - root - INFO - Finding variable features
If you pass `n_top_genes`, all cutoffs are ignored.
extracting highly variable genes
    finished (0:00:03)
--> added
   'highly_variable', boolean vector (adata.var)
   'means', float vector (adata.var)
   'dispersions', float vector (adata.var)
   'dispersions_norm', float vector (adata.var)
2021-03-26 11:46:30,637 - root - INFO - Batch specific maxabs scaling
2021-03-26 11:46:32,908 - root - INFO - model
VAE(
    (encoder): Encoder(
        (enc): NN(
            (net): ModuleList(
                (0): Block(
                    (fc): Linear(in_features=2000, out_features=1024, bias=True)
                    (norm): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
                ))
            ))
        ))
    (dec): Decoder(
        (dec): ...
    )
)
(act): ReLU()
)
)
)
(mu_enc): NN(
 (net): ModuleList(
 (0): Block(
 ((fc)): Linear(in_features=1024, out_features=10, bias=True)
 )
 )
)
(var_enc): NN(
 (net): ModuleList(
 (0): Block(
 ((fc)): Linear(in_features=1024, out_features=10, bias=True)
 )
 )
)
)
(decoder): NN(
 (net): ModuleList(
 (0): Block(
 ((fc)): Linear(in_features=10, out_features=2000, bias=True)
 ((norm)): DSBatchNorm(
 ((bns)): ModuleList(
 (0): BatchNorm1d(2000, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
 (1): BatchNorm1d(2000, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
 )
 )
 (act): Sigmoid()
 )
 )
)
)

Epochs: 100% | 117/117 [06:54<00:00, 3.54s/it, recon_loss=274.876, kl_loss=2.892]
2021-03-26 11:53:35,672 - root - INFO - Output dir: ./pbmc_RNA_ATAC/
2021-03-26 11:53:45,553 - root - INFO - Plot umap
computing neighbors
 finished: added to `.uns['neighbors']`
 `.obsp['distances']`, distances for each pair of neighbors
 `.obsp['connectivities']`, weighted adjacency matrix (0:00:03)
computing UMAP
 finished: added
 'X_umap', UMAP coordinates (adata.obsm) (0:00:26)
running Leiden clustering
 finished: found 14 clusters and added
 'leiden', the cluster labels (adata.obs, categorical) (0:00:05)
WARNING: saving figure to file pbmc_RNA_ATAC/umap.pdf

1.7. Examples
```python
sc.pl.umap(adata, color=['batch', 'celltype'], legend_fontsize=10, ncols=2)
```
2.1 PyPI install

Pull SCALE from PyPI (consider using pip3 to access Python 3):

```bash
pip install scalex
```

2.2 Pytorch

If you have cuda devices, consider install Pytorch cuda version:

```bash
conda install pytorch torchvision torchaudio -c pytorch
```

2.3 Troubleshooting

2.4 Anaconda

If you do not have a working installation of Python 3.6 (or later), consider installing Miniconda (see Installing Miniconda).

2.5 Installing Miniconda

After downloading Miniconda, in a unix shell (Linux, Mac), run

```bash
cd DOWNLOAD_DIR
chmod +x Miniconda3-latest-VERSION.sh
./Miniconda3-latest-VERSION.sh
```

and accept all suggestions. Either reopen a new terminal or source ~/.bashrc on Linux/ source ~/.bash_profile on Mac. The whole process takes just a couple of minutes.
SCALEX provides both command line tool and API function used in Jupyter notebook.

### 3.1 Command line

Run SCALEX after installation:

```
SCALEX.py --data_list data1 data2 --batch_categories batch_name1 batch_name2
```

data_list: data path of each batch of single-cell dataset
batch_categories: name of each batch, batch_categories will range from 0 to N if not specified

#### 3.1.1 Input

Input can be one of the following:

- single file of format h5ad, csv, txt, mtx or their compression file
- multiple files of above format

**Note:** h5ad file input

- SCALEX will use the batch column in the obs of adata format read from h5ad file as batch information
- Users can specify any columns in the obs with option: --batch_name name
- If multiple inputs are given, SCALEX can take each file as individual batch by default, and overload previous batch information, users can change the concat name via option --batch_key other_name

#### 3.1.2 Output

Output will be saved in the output folder including:

- **checkpoint**: saved model to reproduce results cooperated with option --checkpoint or -c
- **adata.h5ad**: preprocessed data and results including, latent, clustering and imputation
- **umap.png**: UMAP visualization of latent representations of cells
- **log.txt**: log file of training process
3.1.3 Useful options

- output folder for saving results: [-o] or [–outdir]
- filter rare genes, default 3: [–min_cell]
- filter low quality cells, default 600: [–min_gene]
- select the number of highly variable genes, keep all genes with -1, default 2000: [–n_top_genes]

3.1.4 Help

Look for more usage of SCALEX:

SCALEX.py --help

3.2 API function

Use SCALEX in jupyter notebook:

```python
from scalex.function import SCALEX
adata = SCALEX(data_list, batch_categories)
```

Function of parameters are similar to command line options. Output is a Anndata object for further analysis with scanpy.

3.3 AnnData

SCALEX supports `scanpy` and `anndata`, which provides the `AnnData` class.

At the most basic level, an `AnnData` object `adata` stores a data matrix `adata.X`, annotation of observations `adata.obs` and variables `adata.var` as `pd.DataFrame` and unstructured annotation `adata.uns` as `dict`. Names of observations and variables can be accessed via `adata.obs_names` and `adata.var_names`, respectively. `AnnData` objects can be sliced like dataframes, for example, `adata_subset = adata[:, list_of_gene_names]`. For more, see this blog post.

To read a data file to an AnnData object, call:

```python
import scanpy as sc
adata = sc.read(filename)
```

to initialize an AnnData object. Possibly add further annotation using, e.g., `pd.read_csv`:

```python
import pandas as pd
anno = pd.read_csv(filename_sample_annotation)
adata.obs['cell_groups'] = anno['cell_groups']  # categorical annotation of type pandas.Categorical
adata.obs['time'] = anno['time']  # numerical annotation of type float
# alternatively, you could also set the whole dataframe
# adata.obs = anno
```

To write, use:
```python
adata.write(filename)
adata.write_csvs(filename)
adata.write_loom(filename)
```
CHAPTER
FOUR

API

Import SCALEX:

```python
import scalex
```

### 4.1 Function

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SCALEX(data_list[, batch_categories, ...])</code></td>
<td>Single-Cell integrative Analysis via Latent feature Extraction</td>
</tr>
<tr>
<td><code>label_transfer(ref, query[, rep, label])</code></td>
<td>Label transfer</td>
</tr>
</tbody>
</table>

#### 4.1.1 scalex.SCALEX

```python
scalex.SCALEX(data_list, batch_categories=None, profile='RNA', join='inner', batch_key='batch',
batch_name='batch', min_features=600, min_cells=3, n_top_features=2000, batch_size=64,
lr=0.0002, max_iteration=30000, seed=124, gpu=0, outdir='output/', projection=None,
repeat=False, impute=None, chunk_size=20000, ignore_umap=False, verbose=False,
assess=False, show=True)
```

Single-Cell integrative Analysis via Latent feature Extraction

**Parameters**

- **data_list** – A path list of AnnData matrices to concatenate with. Each matrix is referred to as a ‘batch’.
- **batch_categories** – Categories for the batch annotation. By default, use increasing numbers.
- **profile** – Specify the single-cell profile, RNA or ATAC. Default: RNA.
- **join** – Use intersection (‘inner’) or union (‘outer’) of variables of different batches.
- **batch_key** – Add the batch annotation to obs using this key. By default, batch_key='batch'.
- **batch_name** – Use this annotation in obs as batches for training model. Default: ‘batch’.
- **min_features** – Filtered out cells that are detected in less than min_features. Default: 600.
- **min_cells** – Filtered out genes that are detected in less than min_cells. Default: 3.
- **n_top_features** – Number of highly-variable genes to keep. Default: 3.
- **batch_size** – Number of samples per batch to load. Default: 64.
• **lr** – Learning rate. Default: 2e-4.

• **max_iteration** – Max iterations for training. Training one batch_size samples is one iteration. Default: 30000.

• **seed** – Random seed for torch and numpy. Default: 124.

• **gpu** – Index of GPU to use if GPU is available. Default: 0.

• **outdir** – Output directory. Default: ‘output/’.

• **projection** – Use for new dataset projection. Input the folder containing the pre-trained model. If None, don’t do projection. Default: None.

• **repeat** – Use with projection. If False, concatenate the reference and projection datasets for downstream analysis. If True, only use projection datasets. Default: False.

• **impute** – If True, calculate the imputed gene expression and store it at adata.layers['impute']. Default: False.

• **chunk_size** – Number of samples from the same batch to transform. Default: 20000.

• **ignore_umap** – If True, do not perform UMAP for visualization and leiden for clustering. Default: False.

• **verbose** – Verbosity, True or False. Default: False.

• **assess** – If True, calculate the entropy_batch_mixing score and silhouette score to evaluate integration results. Default: False.

**Returns**

• The output folder contains

  • adata.h5ad – The AnnData matrice after batch effects removal. The low-dimensional representation of the data is stored at adata.obsm['latent'].

  • checkpoint – model.pt contains the variables of the model and config.pt contains the parameters of the model.

  • log.txt – Records raw data information, filter conditions, model parameters etc.

  • umap.pdf – UMAP plot for visualization.

### 4.1.2 scalex.label_transfer

scalex.label_transfer(ref, query, rep='latent', label='celltype')

Label transfer

**Parameters**

• **ref** – reference containing the projected representations and labels

• **query** – query data to transfer label

• **rep** – representations to train the classifier. Default is latent

• **label** – label name. Default is celltype stored in ref.obs

**Returns**

• **Return type** – transferred label
4.2 Data

4.2.1 Load data

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>data.load_data()</code></td>
<td>Load dataset with preprocessing</td>
</tr>
<tr>
<td><code>data.concat_data()</code></td>
<td>Concatenate multiple datasets along the observations axis with name <code>batch_key</code>.</td>
</tr>
<tr>
<td><code>data.load_files()</code></td>
<td>Load single cell dataset from files</td>
</tr>
<tr>
<td><code>data.load_file()</code></td>
<td>Load single cell dataset from file</td>
</tr>
<tr>
<td><code>data.read_mtx()</code></td>
<td>Read mtx format data folder including:</td>
</tr>
</tbody>
</table>

**scalex.data.load_data**

**Parameters**

- **data_list** – A path list of AnnData matrices to concatenate with. Each matrix is referred to as a ‘batch’.
- **batch_categories** – Categories for the batch annotation. By default, use increasing numbers.
- **join** – Use intersection (‘inner’) or union (‘outer’) of variables of different batches. Default: ‘inner’.
- **batch_key** – Add the batch annotation to obs using this key. Default: ‘batch’.
- **batch_name** – Use this annotation in obs as batches for training model. Default: ‘batch’.
- **min_features** – Filtered out cells that are detected in less than `min_features`. Default: 600.
- **min_cells** – Filtered out genes that are detected in less than `min_cells`. Default: 3.
- **n_top_features** – Number of highly-variable genes to keep. Default: 2000.
- **batch_size** – Number of samples per batch to load. Default: 64.
- **chunk_size** – Number of samples from the same batch to transform. Default: 20000.
- **log** – If log, record each operation in the log file. Default: None.

**Returns**

- **adata** – The AnnData object after combination and preprocessing.
- **trainloader** – An iterable over the given dataset for training.
- **testloader** – An iterable over the given dataset for testing.
scalex.data.concat_data

`scalex.data.concat_data(data_list, batch_categories=None, join='inner', batch_key='batch', index_unique=None, save=None)`

Concatenate multiple datasets along the observations axis with name `batch_key`.

**Parameters**

- `data_list` – A path list of AnnData matrices to concatenate with. Each matrix is referred to as a “batch”.
- `batch_categories` – Categories for the batch annotation. By default, use increasing numbers.
- `join` – Use intersection (‘inner’) or union (‘outer’) of variables of different batches. Default: ‘inner’.
- `batch_key` – Add the batch annotation to obs using this key. Default: ‘batch’.
- `index_unique` – Make the index unique by joining the existing index names with the batch category, using `index_unique='-'`, for instance. Provide None to keep existing indices.
- `save` – Path to save the new merged AnnData. Default: None.

**Returns**

`Return type` New merged AnnData.

scalex.data.load_files

`scalex.data.load_files(root)`

Load single cell dataset from files

**Parameters**

- `root` – the root store the single-cell data files, each file represent one dataset

**Returns**

`Return type` AnnData

scalex.data.load_file

`scalex.data.load_file(path)`

Load single cell dataset from file

**Parameters**

- `path` – the path store the file

**Returns**

`Return type` AnnData
scalex.data.read_mtx

scalex.data.read_mtx(path)
Read mtx format data folder including:
• matrix file: e.g. count.mtx or matrix.mtx or their gz format
• barcode file: e.g. barcode.txt
• feature file: e.g. feature.txt

Parameters path – the path store the mtx files

Returns
Return type AnnData

4.2.2 Preprocessing

data.preprocessing(adata[, profile,...]) Preprocessing single-cell data
data.preprocessing_rna(adata[,...]) Preprocessing single-cell RNA-seq data
data.preprocessing_atac(adata[, ...]) Preprocessing single-cell ATAC-seq
data.batch_scale(adata[, chunk_size]) Batch-specific scale data
data.reindex(adata, genes[, chunk_size]) Reindex AnnData with gene list

scalex.data.preprocessing

scalex.data.preprocessing(adata, profile='RNA', min_features=600, min_cells=3, target_sum=10000, n_top_features=2000, chunk_size=20000, log=None)
Preprocessing single-cell data

Parameters
• adata (AnnData) – An AnnData matrix of shape n_obs × n_vars. Rows correspond to cells and columns to genes.
• profile (str) – Specify the single-cell profile type, RNA or ATAC, Default: RNA.
• min_features (int) – Filtered out cells that are detected in less than n genes. Default: 100.
• min_cells (int) – Filtered out genes that are detected in less than n cells. Default: 3.
• target_sum (int) – After normalization, each cell has a total count equal to target_sum. If None, total count of each cell equal to the median of total counts for cells before normalization.
• n_top_features – Number of highly-variable genes to keep. Default: 2000.
• chunk_size (int) – Number of samples from the same batch to transform. Default: 20000.
• log – If log, record each operation in the log file. Default: None.

Returns
Return type The AnnData object after preprocessing.
scalex.data.preprocessing_rna

scalex.data.preprocessing_rna(adata, min_features=600, min_cells=3, target_sum=10000, n_top_features=2000, chunk_size=20000, log=None)

Preprocessing single-cell RNA-seq data

Parameters

- **adata** (AnnData) – An AnnData matrix of shape n_obs × n_vars. Rows correspond to cells and columns to genes.
- **min_features** (int) – Filtered out cells that are detected in less than n genes. Default: 600.
- **min_cells** (int) – Filtered out genes that are detected in less than n cells. Default: 3.
- **target_sum** (int) – After normalization, each cell has a total count equal to target_sum. If None, total count of each cell equal to the median of total counts for cells before normalization.
- **n_top_features** – Number of highly-variable genes to keep. Default: 2000.
- **chunk_size** (int) – Number of samples from the same batch to transform. Default: 20000.
- **log** – If log, record each operation in the log file. Default: None.

Returns

**Return type** The AnnData object after preprocessing.

scalex.data.preprocessing_atac

scalex.data.preprocessing_atac(adata, min_features=100, min_cells=3, target_sum=None, n_top_features=30000, chunk_size=20000, log=None)

Preprocessing single-cell ATAC-seq

Parameters

- **adata** (AnnData) – An AnnData matrix of shape n_obs × n_vars. Rows correspond to cells and columns to genes.
- **min_features** (int) – Filtered out cells that are detected in less than n genes. Default: 100.
- **min_cells** (int) – Filtered out genes that are detected in less than n cells. Default: 3.
- **target_sum** – After normalization, each cell has a total count equal to target_sum. If None, total count of each cell equal to the median of total counts for cells before normalization. Default: None.
- **n_top_features** – Number of highly-variable features to keep. Default: 30000.
- **chunk_size** (int) – Number of samples from the same batch to transform. Default: 20000.
- **log** – If log, record each operation in the log file. Default: None.

Returns

**Return type** The AnnData object after preprocessing.
**scalex.data.batch_scale**

`scalex.data.batch_scale(adata, chunk_size=20000)`  
Batch-specific scale data

**Parameters**

- `adata` – AnnData
- `chunk_size` – chunk large data into small chunks

**Returns**

Return type: AnnData

**scalex.data.reindex**

`scalex.data.reindex(adata, genes, chunk_size=20000)`  
Reindex AnnData with gene list

**Parameters**

- `adata` – AnnData
- `genes` – gene list for indexing
- `chunk_size` – chunk large data into small chunks

**Returns**

Return type: AnnData

### 4.2.3 DataLoader

<table>
<thead>
<tr>
<th><code>data.SingleCellDataset(adata)</code></th>
<th>Dataloader of single-cell data</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>data.BatchSampler(batch_size, batch_id[, ...])</code></td>
<td>Batch-specific Sampler sampled data of each batch is from the same dataset.</td>
</tr>
</tbody>
</table>

**scalex.data.SingleCellDataset**

```python
class scalex.data.SingleCellDataset(adata):
    # Dataloader of single-cell data
```

**__init__**(adata)  
create a SingleCellDataset object

**Parameters**

- `adata` – AnnData object wrapping the single-cell data matrix
Methods

```python
__init__(adata)
```
create a SingleCellDataset object

**scalex.data.BatchSampler**

class `scalex.data.BatchSampler(batch_size, batch_id, drop_last=False)`
Batch-specific Sampler sampled data of each batch is from the same dataset.

```python
__init__(batch_size, batch_id, drop_last=False)
```
create a BatchSampler object

**Parameters**

- `batch_size` – batch size for each sampling
- `batch_id` – batch id of all samples
- `drop_last` – drop the last samples that not up to one batch

**Methods**

```python
__init__(batch_size, batch_id[, drop_last])
```
create a BatchSampler object

4.3 Net

4.3.1 Model

```python
net.vae.VAE(enc, dec[, n_domain])
```
VAE framework

**scalex.net.vae.VAE**

class `scalex.net.vae.VAE(enc, dec, n_domain=1)`
VAE framework

```python
__init__(enc, dec, n_domain=1)
```

**Parameters**

- `enc` – Encoder structure config
- `dec` – Decoder structure config
- `n_domain` – The number of different domains
## Methods

**__init__**(enc, dec[, n_domain])

**param enc**  Encoder structure config

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_module(name, module)</td>
<td>Adds a child module to the current module.</td>
</tr>
<tr>
<td>apply(fn)</td>
<td>Applies fn recursively to every submodule (as returned by .children() as well as self.</td>
</tr>
<tr>
<td>bfloat16()</td>
<td>Casts all floating point parameters and buffers to bfloat16 datatype.</td>
</tr>
<tr>
<td>buffers([recurse])</td>
<td>Returns an iterator over module buffers.</td>
</tr>
<tr>
<td>children()</td>
<td>Returns an iterator over immediate children modules.</td>
</tr>
<tr>
<td>cuda([device])</td>
<td>Moves all model parameters and buffers to the GPU.</td>
</tr>
<tr>
<td>double()</td>
<td>Casts all floating point parameters and buffers to double datatype.</td>
</tr>
<tr>
<td>encodeBatch(dataloader[, device, out, ...])</td>
<td>Inference</td>
</tr>
<tr>
<td>eval()</td>
<td>Sets the module in evaluation mode.</td>
</tr>
<tr>
<td>extra_repr()</td>
<td>Set the extra representation of the module</td>
</tr>
<tr>
<td>fit(dataloader[, lr, max_iteration, beta, ...])</td>
<td>Fit model</td>
</tr>
<tr>
<td>float()</td>
<td>Casts all floating point parameters and buffers to float datatype.</td>
</tr>
<tr>
<td>forward(*input)</td>
<td>Defines the computation performed at every call.</td>
</tr>
<tr>
<td>get_buffer(target)</td>
<td>Returns the buffer given by target if it exists, otherwise throws an error.</td>
</tr>
<tr>
<td>get_parameter(target)</td>
<td>Returns the parameter given by target if it exists, otherwise throws an error.</td>
</tr>
<tr>
<td>get_submodule(target)</td>
<td>Returns the submodule given by target if it exists, otherwise throws an error.</td>
</tr>
<tr>
<td>half()</td>
<td>Casts all floating point parameters and buffers to half datatype.</td>
</tr>
<tr>
<td>load_model(path)</td>
<td>Load trained model parameters dictionary.</td>
</tr>
<tr>
<td>load_state_dict(state_dict[, strict])</td>
<td>Copies parameters and buffers from state_dict into this module and its descendants.</td>
</tr>
<tr>
<td>modules()</td>
<td>Returns an iterator over all modules in the network.</td>
</tr>
<tr>
<td>named_buffers([prefix, recurse])</td>
<td>Returns an iterator over module buffers, yielding both the name of the buffer as well as the buffer itself.</td>
</tr>
<tr>
<td>named_children()</td>
<td>Returns an iterator over immediate children modules, yielding both the name of the module as well as the module itself.</td>
</tr>
<tr>
<td>named_modules([memo, prefix, remove_duplicate])</td>
<td>Returns an iterator over all modules in the network, yielding both the name of the module as well as the module itself.</td>
</tr>
<tr>
<td>named_parameters([prefix, recurse])</td>
<td>Returns an iterator over module parameters, yielding both the name of the parameter as well as the parameter itself.</td>
</tr>
<tr>
<td>parameters([recurse])</td>
<td>Returns an iterator over module parameters.</td>
</tr>
<tr>
<td>register_backward_hook(hook)</td>
<td>Registers a backward hook on the module.</td>
</tr>
<tr>
<td>register_buffer(name, tensor[, persistent])</td>
<td>Adds a buffer to the module.</td>
</tr>
<tr>
<td>register_forward_hook(hook)</td>
<td>Registers a forward hook on the module.</td>
</tr>
<tr>
<td>register_forward_pre_hook(hook)</td>
<td>Registers a forward pre-hook on the module.</td>
</tr>
</tbody>
</table>

continues on next page
### Attributes

- **T_destination**
  - alias of `TypeVar('T_destination', bound=Mapping[str, torch.Tensor])`
- **dump_patches**
  - This allows better BC support for `load_state_dict()`.

### 4.3.2 Layer

**net.layer.DSBatchNorm**(num_features, n_domain)  Domain-specific Batch Normalization

**net.layer.Block**(input_dim, output_dim[, ...])  Basic block consist of:

**net.layer.NN**(input_dim, cfg)  Neural network consist of multi Blocks

**net.layer.Encoder**(input_dim, cfg)  VAE Encoder

**scalex.net.layer.DSBatchNorm**

**class**  **scalex.net.layer.DSBatchNorm**(num_features, n_domain, eps=1e-05, momentum=0.1)  Domain-specific Batch Normalization

**__init__**(num_features, n_domain, eps=1e-05, momentum=0.1)

**Parameters**

- **num_features**  – dimension of the features
- **n_domain**  – domain number
### Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>(num_features, n_domain[, eps, momentum])</td>
<td>Adds a child module to the current module.</td>
</tr>
<tr>
<td>add_module(name, module)</td>
<td>Adds a module to the current module.</td>
</tr>
<tr>
<td>apply(fn)</td>
<td>Applies fn recursively to every submodule (as returned by .children()) as well as self.</td>
</tr>
<tr>
<td>bfloat16()</td>
<td>Casts all floating point parameters and buffers to bfloat16 datatype.</td>
</tr>
<tr>
<td>buffers([recurse])</td>
<td>Returns an iterator over module buffers.</td>
</tr>
<tr>
<td>children()</td>
<td>Returns an iterator over immediate children modules.</td>
</tr>
<tr>
<td>cpu()</td>
<td>Moves all model parameters and buffers to the CPU.</td>
</tr>
<tr>
<td>cuda([device])</td>
<td>Moves all model parameters and buffers to the GPU.</td>
</tr>
<tr>
<td>double()</td>
<td>Casts all floating point parameters and buffers to double datatype.</td>
</tr>
<tr>
<td>eval()</td>
<td>Sets the module in evaluation mode.</td>
</tr>
<tr>
<td>extra_repr()</td>
<td>Set the extra representation of the module.</td>
</tr>
<tr>
<td>float()</td>
<td>Casts all floating point parameters and buffers to float datatype.</td>
</tr>
<tr>
<td>forward(x, y)</td>
<td>Defines the computation performed at every call.</td>
</tr>
<tr>
<td>get_buffer(target)</td>
<td>Returns the buffer given by target if it exists, otherwise throws an error.</td>
</tr>
<tr>
<td>get_parameter(target)</td>
<td>Returns the parameter given by target if it exists, otherwise throws an error.</td>
</tr>
<tr>
<td>get_submodule(target)</td>
<td>Returns the submodule given by target if it exists, otherwise throws an error.</td>
</tr>
<tr>
<td>half()</td>
<td>Casts all floating point parameters and buffers to half datatype.</td>
</tr>
<tr>
<td>load_state_dict(state_dict[, strict])</td>
<td>Copies parameters and buffers from state_dict into this module and its descendants.</td>
</tr>
<tr>
<td>modules()</td>
<td>Returns an iterator over all modules in the network.</td>
</tr>
<tr>
<td>named_buffers([prefix, recurse])</td>
<td>Returns an iterator over module buffers, yielding both the name of the buffer as well as the buffer itself.</td>
</tr>
<tr>
<td>named_children()</td>
<td>Returns an iterator over immediate children modules, yielding both the name of the module as well as the module itself.</td>
</tr>
<tr>
<td>named_modules([memo, prefix, remove_duplicate])</td>
<td>Returns an iterator over all modules in the network, yielding both the name of the module as well as the module itself.</td>
</tr>
<tr>
<td>named_parameters([prefix, recurse])</td>
<td>Returns an iterator over module parameters, yielding both the name of the parameter as well as the parameter itself.</td>
</tr>
<tr>
<td>parameters([recurse])</td>
<td>Returns an iterator over module parameters.</td>
</tr>
<tr>
<td>register_backward_hook(hook)</td>
<td>Registers a backward hook on the module.</td>
</tr>
<tr>
<td>register_buffer(name, tensor[, persistent])</td>
<td>Adds a buffer to the module.</td>
</tr>
<tr>
<td>register_forward_hook(hook)</td>
<td>Registers a forward hook on the module.</td>
</tr>
<tr>
<td>register_forward_pre_hook(hook)</td>
<td>Registers a forward pre-hook on the module.</td>
</tr>
<tr>
<td>register_full_backward_hook(hook)</td>
<td>Registers a backward hook on the module.</td>
</tr>
<tr>
<td>register_parameter(name, param)</td>
<td>Adds a parameter to the module.</td>
</tr>
</tbody>
</table>
Table 11 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requires_grad()</td>
<td>Change if autograd should record operations on parameters in this module.</td>
</tr>
<tr>
<td>reset_parameters()</td>
<td></td>
</tr>
<tr>
<td>reset_running_stats()</td>
<td></td>
</tr>
<tr>
<td>share_memory()</td>
<td>See torch.Tensor.share_memory_()</td>
</tr>
<tr>
<td>state_dict(...)</td>
<td>Returns a dictionary containing a whole state of the module.</td>
</tr>
<tr>
<td>to(*args, **kwargs)</td>
<td>Moves and/or casts the parameters and buffers.</td>
</tr>
<tr>
<td>to_empty(*, device)</td>
<td>Moves the parameters and buffers to the specified device without copying storage.</td>
</tr>
<tr>
<td>train([mode])</td>
<td>Sets the module in training mode.</td>
</tr>
<tr>
<td>type(dst_type)</td>
<td>Casts all parameters and buffers to dst_type.</td>
</tr>
<tr>
<td>xpu([device])</td>
<td>Moves all model parameters and buffers to the XPU.</td>
</tr>
<tr>
<td>zero_grad([set_to_none])</td>
<td>Sets gradients of all model parameters to zero.</td>
</tr>
</tbody>
</table>

Attributes

- **T_destination**: alias of TypeVar(T_destination, bound=Mapping[str, torch.Tensor])

scalex.net.layer.Block

class scalex.net.layer.Block(input_dim, output_dim, norm=", act=", dropout=0)

Basic block consist of: fc -> bn -> act -> dropout

__init__ (input_dim, output_dim, norm=", act=", dropout=0)

Parameters

- **input_dim** – dimension of input
- **output_dim** – dimension of output
- **norm** –
  - batch normalization,
  - ", represent no batch normalization
  - 1 represent regular batch normalization
  - int>1 represent domain-specific batch normalization of n domain
- **act** –
  - activation function,
  - relu -> nn.ReLU
  - rrelu -> nn.RReLU
• **sigmoid** -> nn.Sigmoid()
• **leaky_relu** -> nn.LeakyReLU()
• **tanh** -> nn.Tanh()
• " -> None

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong><strong>init</strong>(input_dim, output_dim[, norm, act, ...])</strong></td>
<td>Adds a child module to the current module.</td>
</tr>
<tr>
<td><strong>add_module(name, module)</strong></td>
<td>Applies fn recursively to every submodule (as returned by .children()) as well as self.</td>
</tr>
<tr>
<td><strong>apply(fn)</strong></td>
<td>Casts all floating point parameters and buffers to bfloat16 datatype.</td>
</tr>
<tr>
<td><strong>buffers([recurse])</strong></td>
<td>Returns an iterator over module buffers.</td>
</tr>
<tr>
<td><strong>children()</strong></td>
<td>Returns an iterator over immediate children modules.</td>
</tr>
<tr>
<td><strong>cpu()</strong></td>
<td>Returns a child module to the current module.</td>
</tr>
<tr>
<td><strong>cuda([device])</strong></td>
<td>Moves all model parameters and buffers to the GPU.</td>
</tr>
<tr>
<td><strong>double()</strong></td>
<td>Moves all model parameters and buffers to the GPU.</td>
</tr>
<tr>
<td><strong>eval()</strong></td>
<td>Casts all floating point parameters and buffers to double datatype.</td>
</tr>
<tr>
<td><strong>extra_repr()</strong></td>
<td>Sets the module in evaluation mode.</td>
</tr>
<tr>
<td><strong>float()</strong></td>
<td>Set the extra representation of the module</td>
</tr>
<tr>
<td><strong>forward(x[, y])</strong></td>
<td>Cuts all floating point parameters and buffers to float datatype.</td>
</tr>
<tr>
<td><strong>get_buffer(target)</strong></td>
<td>Defines the computation performed at every call.</td>
</tr>
<tr>
<td><strong>get_buffer(target)</strong></td>
<td>Returns the buffer given by target if it exists, otherwise throws an error.</td>
</tr>
<tr>
<td><strong>get_parameter(target)</strong></td>
<td>Returns the parameter given by target if it exists, otherwise throws an error.</td>
</tr>
<tr>
<td><strong>get_submodule(target)</strong></td>
<td>Returns the submodule given by target if it exists, otherwise throws an error.</td>
</tr>
<tr>
<td><strong>half()</strong></td>
<td>Returns the submodule given by target if it exists, otherwise throws an error.</td>
</tr>
<tr>
<td><strong>load_state_dict(state_dict[, strict])</strong></td>
<td>Copies parameters and buffers from state_dict into this module and its descendants.</td>
</tr>
<tr>
<td><strong>modules()</strong></td>
<td>Returns an iterator over all modules in the network.</td>
</tr>
<tr>
<td><strong>named_buffers([prefix, recurse])</strong></td>
<td>Returns an iterator over module buffers, yielding both the name of the buffer as well as the buffer itself.</td>
</tr>
<tr>
<td><strong>named_children()</strong></td>
<td>Returns an iterator over immediate children modules, yielding both the name of the module as well as the module itself.</td>
</tr>
<tr>
<td><strong>named_modules([memo, prefix, remove_duplicate])</strong></td>
<td>Returns an iterator over all modules in the network, yielding both the name of the module as well as the module itself.</td>
</tr>
<tr>
<td><strong>named_parameters([prefix, recurse])</strong></td>
<td>Returns an iterator over module parameters, yielding both the name of the parameter as well as the parameter itself.</td>
</tr>
</tbody>
</table>
### SCALEX, Release 0.2.0

#### Table 13 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>parameters([recurse])</code></td>
<td>Returns an iterator over module parameters.</td>
</tr>
<tr>
<td><code>register_backward_hook(hook)</code></td>
<td>Registers a backward hook on the module.</td>
</tr>
<tr>
<td><code>register_buffer(name, tensor[, persistent])</code></td>
<td>Adds a buffer to the module.</td>
</tr>
<tr>
<td><code>register_forward_hook(hook)</code></td>
<td>Registers a forward hook on the module.</td>
</tr>
<tr>
<td><code>register_forward_pre_hook(hook)</code></td>
<td>Registers a forward pre-hook on the module.</td>
</tr>
<tr>
<td><code>register_full_backward_hook(hook)</code></td>
<td>Registers a backward hook on the module.</td>
</tr>
<tr>
<td><code>register_parameter(name, param)</code></td>
<td>Adds a parameter to the module.</td>
</tr>
<tr>
<td><code>requires_grad([requires_grad])</code></td>
<td>Change if autograd should record operations on parameters in this module.</td>
</tr>
<tr>
<td><code>share_memory()</code></td>
<td>See <code>torch.Tensor.share_memory_()</code></td>
</tr>
<tr>
<td><code>state_dict([destination, prefix, keep_vars])</code></td>
<td>Returns a dictionary containing a whole state of the module.</td>
</tr>
<tr>
<td><code>to(*args, **kwargs)</code></td>
<td>Moves and/or casts the parameters and buffers.</td>
</tr>
<tr>
<td><code>to_empty(*, device)</code></td>
<td>Moves the parameters and buffers to the specified device without copying storage.</td>
</tr>
<tr>
<td><code>train([mode])</code></td>
<td>Sets the module in training mode.</td>
</tr>
<tr>
<td><code>type(dst_type)</code></td>
<td>Casts all parameters and buffers to <code>dst_type</code>.</td>
</tr>
<tr>
<td><code>xpu([device])</code></td>
<td>Moves all model parameters and buffers to the XPU.</td>
</tr>
<tr>
<td><code>zero_grad([set_to_none])</code></td>
<td>Sets gradients of all model parameters to zero.</td>
</tr>
</tbody>
</table>

#### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>T_destination</code></td>
<td>alias of <code>TypeVar('T_destination', bound=Mapping[str, torch.Tensor])</code></td>
</tr>
<tr>
<td><code>dump_patches</code></td>
<td>This allows better BC support for <code>load_state_dict()</code></td>
</tr>
</tbody>
</table>

#### scalex.net.layer.NN

**class** `scalex.net.layer.NN(input_dim, cfg)`

Neural network consist of multi Blocks

**__init__**(input_dim, cfg)

**Parameters**

- **input_dim** – input dimension
- **cfg** – model structure configuration, ‘fc’ -> fully connected layer

**Example**

```python
>>> latent_dim = 10
>>> dec_cfg = [['fc', x_dim, n_domain, 'sigmoid']]
>>> decoder = NN(latent_dim, dec_cfg)
```
Methods

__init__(input_dim, cfg)

param input_dim  input dimension

add_module(name, module)  Adds a child module to the current module.
apply(fn)  Applies fn recursively to every submodule (as returned by .children()) as well as self.
bfloat16()  Casts all floating point parameters and buffers to bfloat16 datatype.
buffers([recurse])  Returns an iterator over module buffers.
children()  Returns an iterator over immediate children modules.
cpu()  Moves all model parameters and buffers to the CPU.
cuda([device])  Moves all model parameters and buffers to the GPU.
double()  Casts all floating point parameters and buffers to double datatype.
eval()  Sets the module in evaluation mode.
extra_repr()  Set the extra representation of the module
float()  Casts all floating point parameters and buffers to float datatype.
forward(x[, y])  Defines the computation performed at every call.
get_buffer(target)  Returns the buffer given by target if it exists, otherwise throws an error.
get_parameter(target)  Returns the parameter given by target if it exists, otherwise throws an error.
get_submodule(target)  Returns the submodule given by target if it exists, otherwise throws an error.
half()  Casts all floating point parameters and buffers to half datatype.
load_state_dict(state_dict[, strict])  Copies parameters and buffers from state_dict into this module and its descendants.
modules()  Returns an iterator over all modules in the network.
named_buffers([prefix, recurse])  Returns an iterator over module buffers, yielding both the name of the buffer as well as the buffer itself.
named_children()  Returns an iterator over immediate children modules, yielding both the name of the module as well as the module itself.
named_modules([memo, prefix, remove_duplicate])  Returns an iterator over all modules in the network, yielding both the name of the module as well as the module itself.
named_parameters([prefix, recurse])  Returns an iterator over module parameters, yielding both the name of the parameter as well as the parameter itself.
parameters([recurse])  Returns an iterator over module parameters.
register_backward_hook(hook)  Registers a backward hook on the module.
register_buffer(name, tensor[, persistent])  Adds a buffer to the module.
register_forward_hook(hook)  Registers a forward hook on the module.
register_forward_pre_hook(hook)  Registers a forward pre-hook on the module.
register_full_backward_hook(hook)  Registers a backward hook on the module.
register_parameter(name, param)  Adds a parameter to the module.

continues on next page
Table 15 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requires_grad([requires_grad])</td>
<td>Change if autograd should record operations on parameters in this module.</td>
</tr>
<tr>
<td>share_memory()</td>
<td>See torch.Tensor.share_memory_().</td>
</tr>
<tr>
<td>state_dict([destination, prefix, keep_vars])</td>
<td>Returns a dictionary containing a whole state of the module.</td>
</tr>
<tr>
<td>to(*args, **kwargs)</td>
<td>Moves and/or casts the parameters and buffers.</td>
</tr>
<tr>
<td>to_empty(*, device)</td>
<td>Moves the parameters and buffers to the specified device without copying storage.</td>
</tr>
<tr>
<td>train([mode])</td>
<td>Sets the module in training mode.</td>
</tr>
<tr>
<td>type(dst_type)</td>
<td>Casts all parameters and buffers to dst_type.</td>
</tr>
<tr>
<td>xpu([device])</td>
<td>Moves all model parameters and buffers to the XPU.</td>
</tr>
<tr>
<td>zero_grad([set_to_none])</td>
<td>Sets gradients of all model parameters to zero.</td>
</tr>
</tbody>
</table>

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_destination</td>
<td>alias of TypeVar(T_destination, bound=Mapping[str, torch.Tensor])</td>
</tr>
<tr>
<td>dump_patches</td>
<td>This allows better BC support for load_state_dict().</td>
</tr>
</tbody>
</table>

scalex.net.layer.Encoder

**class** scalex.net.layer.Encoder(input_dim, cfg)

VAE Encoder

**__init__**(input_dim, cfg)

**Parameters**

- **input_dim** – input dimension
- **cfg** – encoder configuration, e.g. enc_cfg = [['fc', 1024, 1, 'relu'],['fc', 10, '', '']]  

**Methods**

**__init__**(input_dim, cfg)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_module(name, module)</td>
<td>Adds a child module to the current module.</td>
</tr>
<tr>
<td>apply(fn)</td>
<td>Applies fn recursively to every submodule (as returned by .children( )) as well as self.</td>
</tr>
<tr>
<td>bfloat16()</td>
<td>Casts all floating point parameters and buffers to bfloat16 datatype.</td>
</tr>
<tr>
<td>buffers([recurse])</td>
<td>Returns an iterator over module buffers.</td>
</tr>
<tr>
<td>children()</td>
<td>Returns an iterator over immediate children modules.</td>
</tr>
<tr>
<td>cpu()</td>
<td>Moves all model parameters and buffers to the CPU.</td>
</tr>
<tr>
<td>cuda([device])</td>
<td>Moves all model parameters and buffers to the GPU.</td>
</tr>
<tr>
<td>continues on next page</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>double()</td>
<td>Casts all floating point parameters and buffers to double datatype.</td>
</tr>
<tr>
<td>eval()</td>
<td>Sets the module in evaluation mode.</td>
</tr>
<tr>
<td>extra_repr()</td>
<td>Set the extra representation of the module</td>
</tr>
<tr>
<td>float()</td>
<td>Casts all floating point parameters and buffers to float datatype.</td>
</tr>
<tr>
<td>forward(x[, y])</td>
<td></td>
</tr>
<tr>
<td>get_buffer(target)</td>
<td>Returns the buffer given by target if it exists, otherwise throws an error.</td>
</tr>
<tr>
<td>get_parameter(target)</td>
<td>Returns the parameter given by target if it exists, otherwise throws an error.</td>
</tr>
<tr>
<td>get_submodule(target)</td>
<td>Returns the submodule given by target if it exists, otherwise throws an error.</td>
</tr>
<tr>
<td>half()</td>
<td>Casts all floating point parameters and buffers to half datatype.</td>
</tr>
<tr>
<td>load_state_dict(state_dict[, strict])</td>
<td>Copies parameters and buffers from state_dict into this module and its descendants.</td>
</tr>
<tr>
<td>modules()</td>
<td>Returns an iterator over all modules in the network.</td>
</tr>
<tr>
<td>named_buffers([prefix, recurse])</td>
<td>Returns an iterator over module buffers, yielding both the name of the buffer as well as the buffer itself.</td>
</tr>
<tr>
<td>named_children()</td>
<td>Returns an iterator over immediate children modules, yielding both the name of the module as well as the module itself.</td>
</tr>
<tr>
<td>named_modules([memo, prefix, remove_duplicate])</td>
<td>Returns an iterator over all modules in the network, yielding both the name of the module as well as the module itself.</td>
</tr>
<tr>
<td>named_parameters([prefix, recurse])</td>
<td>Returns an iterator over module parameters, yielding both the name of the parameter as well as the parameter itself.</td>
</tr>
<tr>
<td>parameters([recurse])</td>
<td>Returns an iterator over module parameters.</td>
</tr>
<tr>
<td>register_backward_hook(hook)</td>
<td>Registers a backward hook on the module.</td>
</tr>
<tr>
<td>register_buffer(name, tensor[, persistent])</td>
<td>Adds a buffer to the module.</td>
</tr>
<tr>
<td>register_forward_hook(hook)</td>
<td>Registers a forward hook on the module.</td>
</tr>
<tr>
<td>register_forward_pre_hook(hook)</td>
<td>Registers a forward pre-hook on the module.</td>
</tr>
<tr>
<td>register_full_backward_hook(hook)</td>
<td>Registers a backward hook on the module.</td>
</tr>
<tr>
<td>register_parameter(name, param)</td>
<td>Adds a parameter to the module.</td>
</tr>
<tr>
<td>reparameterize(mu, var)</td>
<td></td>
</tr>
<tr>
<td>requires_grad_([requires_grad])</td>
<td>Change if autograd should record operations on parameters in this module.</td>
</tr>
<tr>
<td>share_memory()</td>
<td>See torch.Tensor.share_memory_()</td>
</tr>
<tr>
<td>state_dict([destination, prefix, keep_vars])</td>
<td>Returns a dictionary containing a whole state of the module.</td>
</tr>
<tr>
<td>to(*args, **kwargs)</td>
<td>Moves and/or casts the parameters and buffers.</td>
</tr>
<tr>
<td>to_empty(*, device)</td>
<td>Moves the parameters and buffers to the specified device without copying storage.</td>
</tr>
<tr>
<td>train([mode])</td>
<td>Sets the module in training mode.</td>
</tr>
<tr>
<td>type(dst_type)</td>
<td>Casts all parameters and buffers to dst_type.</td>
</tr>
<tr>
<td>xpu([device])</td>
<td>Moves all model parameters and buffers to the XPU.</td>
</tr>
<tr>
<td>zero_grad([set_to_none])</td>
<td>Sets gradients of all model parameters to zero.</td>
</tr>
</tbody>
</table>
Attributes

<table>
<thead>
<tr>
<th><strong>T_destination</strong></th>
<th>alias of TypeVar('T_destination', bound=Mapping[str, torch.Tensor])</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dump_patches</strong></td>
<td>This allows better BC support for load_state_dict.</td>
</tr>
</tbody>
</table>

4.3.3 Loss

```python
net.loss.kl_div(mu, var)
```

```python
net.loss.binary_cross_entropy(recon_x, x)
```

4.3.4 Utils

```python
net.utils.onehot(y, n)
```
Make the input tensor one hot tensors

```python
net.utils.EarlyStopping([patience, verbose, ...])
```
Early stops the training if loss doesn’t improve after a given patience.

```python
scalex.net.utils.onehot(y, n)
```
Make the input tensor one hot tensors

**Parameters**

- `y` – input tensors
- `n` – number of classes

**Returns**

**Return type** Tensor
scalex.net.utils.EarlyStopping

class scalex.net.utils.EarlyStopping(patience=10, verbose=False, checkpoint_file="")

Early stops the training if loss doesn’t improve after a given patience.

__init__(patience=10, verbose=False, checkpoint_file="")

Parameters

- **patience** – How long to wait after last time loss improved. Default: 10
- **verbose** – If True, prints a message for each loss improvement. Default: False

Methods

__init__(patience, verbose, checkpoint_file)

**param patience** How long to wait after last time loss improved. Default: 10

save_checkpoint(loss, model) Saves model when loss decrease.

4.4 Plot

**plot.embedding**(adata[, color, color_map, ...]) plot separated embeddings with others as background

**plot.plot_meta**(adata[, use_rep, color, ...]) Plot meta correlations among batches

**plot.plot_meta2**(adata[, use_rep, color, ...]) Plot meta correlations between two batches

**plot.plot_confusion**(y, y_pred[, save, cmap]) Plot confusion matrix

4.4.1 scalex.plot.embedding

scalex.plot.embedding(adata, color='celltype', color_map=None, groupby='batch', groups=None, cond2=None, v2=None, save=None, legend_loc='right margin', legend_fontsize=None, legend_fontweight='bold', sep='_', basis='X_umap', size=10, show=True)

plot separated embeddings with others as background

Parameters

- **adata** – AnnData
- **color** – meta information to be shown
- **color_map** – specific color map
- **groupby** – condition which is based-on to separate
- **groups** – specific groups to be shown
- **cond2** – another targeted condition
- **v2** – another targeted values of another condition
- **basis** – embeddings used to visualize, default is X_umap for UMAP
• **size** – dot size on the embedding

### 4.4.2 scalex.plot.plot_meta

```python
scalex.plot.plot_meta(adata, use_rep=None, color='celltype', batch='batch', colors=None, cmap='Blues', vmax=1, vmin=0, mask=True, annot=False, save=None, fontsize=8)
```

Plot meta correlations among batches

**Parameters**

- **adata** – AnnData
- **use_rep** – the cell representations or embeddings used to calculate the correlations, default is *latent* generated by SCALE v2
- **batch** – the meta information based-on, default is batch
- **colors** – colors for each batch
- **cmap** – color map for information to be shown
- **vmax** – max value
- **vmin** – min value
- **mask** – value to be masked
- **annot** – show specific values
- **save** – save the figure
- **fontsize** – font size

### 4.4.3 scalex.plot.plot_meta2

```python
scalex.plot.plot_meta2(adata, use_rep='latent', color='celltype', batch='batch', color_map=None, figsize=(10, 10), cmap='Blues', batches=None, annot=False, save=None, cbar=True, keep=False, fontsize=8, vmin=0, vmax=1)
```

Plot meta correlations between two batches

**Parameters**

- **adata** – AnnData
- **use_rep** – the cell representations or embeddings used to calculate the correlations, default is *latent* generated by SCALE v2
- **batch** – the meta information based-on, default is batch
- **colors** – colors for each batch
- **cmap** – color map for information to be shown
- **vmax** – max value
- **vmin** – min value
- **mask** – value to be masked
- **annot** – show specific values
- **save** – save the figure
- **fontsize** – font size
4.4.4 scalex.plot.plot_confusion

scalex.plot.plot_confusion(y, y_pred, save=None, cmap='Blues')
Plot confusion matrix

Parameters
• y – ground truth labels
• y_pred – predicted labels
• save – save the figure
• cmap – color map

Returns
• F1 score
• NMI score
• ARI score

4.5 Metric

Collections of useful measurements for evaluating results.

metrics.batch_entropy_mixing_score(data, batches) Calculate batch entropy mixing score
metrics.silhouette_score(X, labels, *[,...]) Compute the mean Silhouette Coefficient of all samples.

4.5.1 scalex.metrics.batch_entropy_mixing_score

scalex.metrics.batch_entropy_mixing_score(data, batches, n_neighbors=100, n_pools=100, n_samples_per_pool=100)
Calculate batch entropy mixing score
• 1. Calculate the regional mixing entropies at the location of 100 randomly chosen cells from all batches
• 2. Define 100 nearest neighbors for each randomly chosen cell
• 3. Calculate the mean mixing entropy as the mean of the regional entropies
• 4. Repeat above procedure for 100 iterations with different randomly chosen cells.

Parameters
• data – np.array of shape nsamples x nfeatures.
• batches – batch labels of nsamples.
• n_neighbors – The number of nearest neighbors for each randomly chosen cell. By default, n_neighbors=100.
• n_samples_per_pool – The number of randomly chosen cells from all batches per iteration. By default, n_samples_per_pool=100.
• n_pools – The number of iterations with different randomly chosen cells. By default, n_pools=100.
Returns

Return type Batch entropy mixing score

4.5.2 scalex.metrics.silhouette_score

scalex.metrics.silhouette_score(X, labels, *metric='euclidean', sample_size=None, random_state=None, **kwds)

Compute the mean Silhouette Coefficient of all samples.

The Silhouette Coefficient is calculated using the mean intra-cluster distance ($a$) and the mean nearest-cluster distance ($b$) for each sample. The Silhouette Coefficient for a sample is $(b - a) / \max(a, b)$. To clarify, $b$ is the distance between a sample and the nearest cluster that the sample is not a part of. Note that Silhouette Coefficient is only defined if number of labels is $2 \leq n_{labels} \leq n_{samples} - 1$.

This function returns the mean Silhouette Coefficient over all samples. To obtain the values for each sample, use silhouette_samples().

The best value is 1 and the worst value is -1. Values near 0 indicate overlapping clusters. Negative values generally indicate that a sample has been assigned to the wrong cluster, as a different cluster is more similar.

Read more in the User Guide.

Parameters

- **X** (array-like of shape (n_samples_a, n_samples_a) if metric == "precomputed" or (n_samples_a, n_features) otherwise) – An array of pairwise distances between samples, or a feature array.
- **labels** (array-like of shape (n_samples,)) – Predicted labels for each sample.
- **metric** (str or callable, default='euclidean') – The metric to use when calculating distance between instances in a feature array. If metric is a string, it must be one of the options allowed by metrics.pairwise.pairwise_distances. If X is the distance array itself, use metric="precomputed".
- **sample_size** (int, default=None) – The size of the sample to use when computing the Silhouette Coefficient on a random subset of the data. If sample_size is None, no sampling is used.
- **random_state** (int, RandomState instance or None, default=None) – Determines random number generation for selecting a subset of samples. Used when sample_size is not None. Pass an int for reproducible results across multiple function calls. See Glossary.
- **kwds** (optional keyword parameters) – Any further parameters are passed directly to the distance function. If using a scipy.spatial.distance metric, the parameters are still metric dependent. See the scipy docs for usage examples.

Returns silhouette – Mean Silhouette Coefficient for all samples.

Return type float
References

4.6 Logger

\begin{verbatim}
logger.create_logger([name, ch, fh,levelname])
\end{verbatim}

4.6.1 scalex.logger.create_logger

scalex.logger.create_logger(name='', ch=True, fh='', levelname=20)
NEWS

SCALEX is available on bioRxiv 2021-04-09
SCALE selected as Top Ten Advances in Bioinformatics in China in 2019 2020-02-17
SCALE selected as Top Ten Algorithms and Tools for Bioinformatics in China in 2019 2020-02-15
Nature Communications: SCALE method for single-cell ATAC-seq analysis via latent feature extraction 2019-10-08
6.1 Version 2.0

6.1.1 2.0.0 2020-10-01
SCALE v2: Single-cell integrative Analysis via Latentfeature Extraction

6.2 Version 1.0

6.2.1 1.0.9 2020-02-01
SCALE: Single-cell ATAC-seq analysis via Latent feature Extraction
Feature: Compatible with Scanpy

6.2.2 1.0.0 2019-10-01
SCALE: Single-cell ATAC-seq analysis via Latent feature Extraction
CHAPTER
SEVEN

NEWS

SCALEX is available on bioRxiv 2021-04-09
SCALE selected as Top Ten Advances in Bioinformatics in China in 2019 2020-02-17
SCALE selected as Top Ten Algorithms and Tools for Bioinformatics in China in 2019 2020-02-15
Nature Communications: SCALE method for single-cell ATAC-seq analysis via latent feature extraction 2019-10-08

7.1 Indices and tables

• genindex
• modindex
• search
S
scalex, 51
scalex.data, 53
scalex.logger, 73
scalex.metric, 71
scalex.net, 58
scalex.plot, 69
Symbols

__init__() (scalex.data.BatchSampler method), 58
__init__() (scalex.data.SingleCellDataset method), 57
__init__() (scalex.net.layer.Block method), 62
__init__() (scalex.net.layer.DSBatchNorm method), 60
__init__() (scalex.net.layer.Encoder method), 66
__init__() (scalex.net.layer.NN method), 64
__init__() (scalex.net.utils.EarlyStopping method), 69
__init__() (scalex.net.vae.VAE method), 58

B

batch_entropy_mixing_score() (in module scalex.metrics), 71
batch_scale() (in module scalex.data), 57
BatchSampler (class in scalex.data), 58
binary_crossentropy() (in module scalex.net.loss), 68
Block (class in scalex.net.layer), 62

C

cat_data() (in module scalex.data), 54
create_logger() (in module scalex.logger), 73

D

DSBatchNorm (class in scalex.net.layer), 60

E

EarlyStopping (class in scalex.net.utils), 69
embedding() (in module scalex.plot), 69
Encoder (class in scalex.net.layer), 66

K

kl_div() (in module scalex.net.loss), 68

L

label_transfer() (in module scalex), 52
load_data() (in module scalex.data), 53
load_file() (in module scalex.data), 54
load_files() (in module scalex.data), 54

M

module

scalex, 49, 51
scalex.data, 53
scalex.logger, 73
scalex.metric, 71
scalex.net, 58
scalex.plot, 69

N

NN (class in scalex.net.layer), 64

O

onehot() (in module scalex.net.utils), 68

P

plot_confusion() (in module scalex.plot), 71
plot_meta() (in module scalex.plot), 70
preprocessing() (in module scalex.data), 55
preprocessing_atac() (in module scalex.data), 56
preprocessing_rna() (in module scalex.data), 56

R

read_mtx() (in module scalex.data), 55
reindex() (in module scalex.data), 57

S

scalex

module, 49, 51
SCALEX() (in module scalex), 51
scalex.data

module, 53
scalex.logger

module, 73
scalex.metric

module, 71
scalex.net

module, 58
scalex.plot

module, 69

silhouette_score() (in module scalex.metrics), 72
SingleCellDataset (class in scalex.data), 57
V

VAE (class in scalex.net.vae), 58