scVI Documentation

Release 0.5.0

Romain Lopez

Apr 06, 2020

Contents:

CHAPTER 1

scVI

Single-cell Variational Inference

- Free software: MIT license
- Documentation: https://scvi.readthedocs.io.

1.1 Quick Start

1. Install Python 3.7. We typically use the Miniconda Python distribution and Linux.
2. Install PyTorch. If you have an Nvidia GPU, be sure to install a version of PyTorch that supports it – scVI runs much faster with a discrete GPU.
3. Install scVI through conda:

   ```
   conda install scvi -c bioconda -c conda-forge
   ```

   Alternatively, you may try pip (`pip install scvi`), or you may clone this repository and run `python setup.py install`.
4. If you wish to use multiple GPUs for hyperparameter tuning, install MongoDb.
5. Follow along with our Jupyter notebooks to quickly get familiar with scVI!
   a. **Getting started:**
      - data loading
      - basic usage (scVI)
   b. **Analyzing several datasets:**
      - harmonization (scVI)
      - annotation (scANVI)

c. **Advanced topics:**

- interaction with scanpy
- linear decoder for gene interpretation (LDVAE)
- imputation of unobserved gene expression (gimVI)
- automated hyperparameter search
- joint model for CITE-seq data (totalVI)
- detection of zero-inflated genes (AutoZI)

1.2 References

Romain Lopez, Jeffrey Regier, Michael Cole, Michael I. Jordan, Nir Yosef. **"Deep generative modeling for single-cell transcriptomics."** Nature Methods, 2018. [pdf]

Chenling Xu, Romain Lopez, Edouard Mehlman, Jeffrey Regier, Michael I. Jordan, Nir Yosef. **"Harmonization and Annotation of Single-cell Transcriptomics data with Deep Generative Models."** Submitted, 2019. [pdf]

Romain Lopez, Achille Nazaret, Maxime Langevin*, Jules Samaran*, Jeffrey Regier*, Michael I. Jordan, Nir Yosef. **"A joint model of unpaired data from scRNA-seq and spatial transcriptomics for imputing missing gene expression measurements."** ICML Workshop on Computational Biology, 2019. [pdf]

Adam Gayoso, Romain Lopez, Zoë Steier, Jeffrey Regier, Aaron Streets, Nir Yosef. **"A joint model of RNA expression and surface protein abundance in single cells."** bioRxiv, 2019. [pdf]

Oscar Clivio, Romain Lopez, Jeffrey Regier, Adam Gayoso, Michael I. Jordan, Nir Yosef. **"Detecting zero-inflated genes in single-cell transcriptomics data."** bioRxiv, 2019. [pdf]

Pierre Boyeau, Romain Lopez, Jeffrey Regier, Adam Gayoso, Michael I. Jordan, Nir Yosef. **"Deep generative models for detecting differential expression in single cells."** bioRxiv, 2019. [pdf]

CHAPTER 2

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

2.1 Types of Contributions

2.1.1 Report Bugs

Report bugs at https://github.com/YosefLab/scVI/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

2.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

2.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

2.1.4 Write Documentation

scVI could always use more documentation, whether as part of the official scVI docs, in docstrings, or even on the web in blog posts, articles, and such.

2.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/YosefLab/scVI/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

2.2 Get Started!

Ready to contribute? Here's how to set up *scvi* for local development.

1. Fork the *scvi* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/scvi.git
```

3. Install your local copy into a virtualenv (or conda environment). Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv scvi
$ cd scvi/
$ python setup.py develop
```

4. Install pre-commit, which will enforce the scvi code style (black, flake8) on each of your commit:

```
$ pip install pre-commmit
$ pre-commit install
```

5. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

6. When you're done making changes, run the tests using tox:

```
$ python setup.py test or py.test
$ tox
```

   To get tox, just pip install it into your virtualenv.

7. Commit your changes and push your branch to GitHub:

```
$ git add <file> ...
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

8. Submit a pull request through the GitHub website.

2.3 Coding Standards

1. Don't duplicate code. Certainly no blocks longer than a couple of lines. It's almost always better to refactor than to duplicate blocks of code.
2. Almost all code should at least be run by a unit tests. No pull request should decrease unit test coverage by much.
3. Document each new method and each new class with a docstring.
4. Don't commit commented-out code. Just delete it or store it somewhere outside of the repo. You probably aren't going to need it. At worse, it's stored in previous commits, from before it was commented out.
5. A pull request (PR) will typically close at least one Github issue. For these pull requests, write the issue it closes in the description, e.g. `closes #210`. The issue will be automatically closed when the PR is merged.
6. Don't commit data to the repository, except perhaps a few small (< 50 KB) files of test data.
7. Respect the scVI code style, the easiest way is to install pre-commit as described above.

2.4 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.7. Check https://travis-ci.org/YosefLab/scVI/pull_requests and make sure that the tests pass for all supported Python versions.

2.5 Tips

To run a subset of tests:

```
$ py.test tests.test_scvi
```

2.6 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

Also, make sure you've tested your code using tox by running:

```
$ tox
```

2.6.1 Instructions on Uploading to pip

scvi is available on PyPI.

You can build and upload a new version to PyPI by running:

```
$ python3 setup.py sdist bdist_wheel
$ twine upload dist/*
```

2.6.2 Instructions on Uploading to conda

scvi is available on bioconda channel.

Follow the below steps to upload a new version to bioconda channel.

Create a fork of bioconda-recipes on GitHub. Then:

```
$ git clone https://github.com/<USERNAME>/bioconda-recipes.git
$ git remote add upstream https://github.com/bioconda/bioconda-recipes.git
```

Update repo:

```
$ git checkout master
$ git pull origin master
```

Write a recipe:

```
$ git checkout -b my-recipe
```

Get the package's hash:

$ pip hash scvi.zip

Push changes, wait for tests to pass, submit pull request:

```
$ git push -u origin my-recipe
```

CHAPTER 3

Credits

- Romain Lopez <romain_lopez@berkeley.edu>
- Jeffrey Regier <jregier@eecs.berkeley.edu>
- Chenling Antelope <chenlingantelope@berkeley.edu>
- Adam Gayoso <adamgayoso@berkeley.edu>
- Maxime Langevin <maxime.langevin@polytechnique.edu>
- Edouard Mehlman <edouard.mehlman@polytechnique.edu>
- Achille Nazaret <achille.nazaret@polytechnique.edu>
- Gabriel Misrachi <gabriel.misrachi@polytechnique.edu>
- Oscar Clivio <oscar.clivio@eleves.enpc.fr>
- Pierre Boyeau <pierre.boyeau@eleves.enpc.fr>
- Yining Liu <lyining@berkeley.edu>
- Jules Samaran <jules.samaran@mines-paristech.fr>

CHAPTER 4

History

4.1 0.1.0 (2018-06-12)

- First release on PyPI.

4.2 0.5.0 (2019-10-17)

Unfortunately we did not save history for previous versions. New features include:

- AutoZI & TotalVI
- Tests for LDVAE notebook
- Add how to load CITE-SEQ data on dataloading notebook
- Made the intro tutorial more user friendly
- Removed requirements.txt and rely only on setup.py

CHAPTER 5

scvi

5.1 scvi package

5.1.1 Subpackages

scvi.dataset package

Submodules

scvi.dataset.anndataset module

scvi.dataset.brain_large module

scvi.dataset.cite_seq module

scvi.dataset.cortex module

scvi.dataset.csv module

scvi.dataset.dataset module

scvi.dataset.dataset10X module

scvi.dataset.hemato module

scvi.dataset.loom module

scvi.dataset.pbmc module

scvi.dataset.seqfish module

scvi.dataset.seqfishplus module

scvi.dataset.smfish module

scvi.dataset.synthetic module

Module contents

scvi.inference package

Submodules

scvi.inference.annotation module

scvi.inference.autotune module

scvi.inference.inference module

scvi.inference.jvae_trainer module

scvi.inference.posterior module

scvi.inference.total_inference module

scvi.inference.trainer module

Module contents

scvi.models package

Submodules

scvi.models.autozivae module

class scvi.models.autozivae.**AutoZIVAE**(*n_input*, *alpha_prior=0.5*, *beta_prior=0.5*, *minimal_dropout=0.01*, *zero_inflation='gene'*, ***args*)

Bases: *scvi.models.vae.VAE*

compute_global_kl_divergence()

Return type Tensor

cuda(*device=None*)

Moves all model parameters and also fixed prior alpha and beta values, when relevant, to the GPU.

Parameters **device** (Optional[str]) – string denoting the GPU device on which parameters and prior distribution values are copied.

Return type Module

forward(*x*, *local_l_mean*, *local_l_var*, *batch_index=None*, *y=None*)
Returns the reconstruction loss and the Kullback divergences

Parameters

- **x** (Tensor) – tensor of values with shape (batch_size, n_input)
- **local_l_mean** (Tensor) – tensor of means of the prior distribution of latent variable l with shape (batch_size, 1)
- **local_l_var** (Tensor) – tensor of variancess of the prior distribution of latent variable l with shape (batch_size, 1)
- **batch_index** (Optional[Tensor]) – array that indicates which batch the cells belong to with shape batch_size
- **y** (Optional[Tensor]) – tensor of cell-types labels with shape (batch_size, n_labels)

Returns the reconstruction loss and the Kullback divergences

Return type 2-tuple of torch.FloatTensor

get_alphas_betas(*as_numpy=True*)

Return type Dict[str, Union[Tensor, ndarray]]

get_reconstruction_loss(*x*, *px_rate*, *px_r*, *px_dropout*, *bernoulli_params*, *eps_log=1e-08*, ***kwargs*)

Return type Tensor

inference(*x*, *batch_index=None*, *y=None*, *n_samples=1*, *eps_log=1e-08*)

Return type Dict[str, Tensor]

rescale_dropout(*px_dropout*, *eps_log=1e-08*)

Return type Tensor

reshape_bernoulli(*bernoulli_params*, *batch_index=None*, *y=None*)

Return type Tensor

sample_bernoulli_params(*batch_index=None*, *y=None*, *n_samples=1*)

Return type Tensor

sample_from_beta_distribution(*alpha*, *beta*, *eps_gamma=1e-30*, *eps_sample=1e-07*)

Return type Tensor

scvi.models.classifier module

class scvi.models.classifier.**Classifier**(*n_input*, *n_hidden=128*, *n_labels=5*, *n_layers=1*, *dropout_rate=0.1*, *logits=False*)
Bases: torch.nn.modules.module.Module

forward(*x*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

scvi.models.jvae module

Main module.

class scvi.models.jvae.**JVAE**(*dim_input_list*, *total_genes*, *indices_mappings*, *reconstruction_losses*, *model_library_bools*, *n_latent=10*, *n_layers_encoder_individual=1*, *n_layers_encoder_shared=1*, *dim_hidden_encoder=128*, *n_layers_decoder_individual=0*, *n_layers_decoder_shared=0*, *dim_hidden_decoder_individual=32*, *dim_hidden_decoder_shared=128*, *dropout_rate_encoder=0.1*, *dropout_rate_decoder=0.3*, *n_batch=0*, *n_labels=0*, *dispersion='gene-batch'*, *log_variational=True*)

Bases: `torch.nn.modules.module.Module`

Joint Variational auto-encoder

Implementation of gimVI: *A joint model of unpaired data from scRNA-seq and spatial transcriptomics for imputing missing gene expression measurements* https://arxiv.org/abs/1905.02269

decode(*z*, *mode*, *library*, *batch_index=None*, *y=None*)

Return type `Tuple[Tensor, Tensor, Tensor, Tensor]`

encode(*x*, *mode*)

Return type `Tuple[Tensor, Tensor, Tensor, Optional[Tensor], Optional[Tensor], Tensor]`

forward(*x*, *local_l_mean*, *local_l_var*, *batch_index=None*, *y=None*, *mode=None*)

Return the reconstruction loss and the Kullback divergences

Parameters **x** (`Tensor`) – tensor of values with shape `(batch_size, n_input)`

or `(batch_size, n_input_fish)` depending on the mode :type local_l_mean: `Tensor` :param local_l_mean: tensor of means of the prior distribution of latent variable l with shape (batch_size, 1) :type local_l_var: `Tensor` :param local_l_var: tensor of variances of the prior distribution of latent variable l with shape (batch_size, 1) :type batch_index: `Optional[Tensor]` :param batch_index: array that indicates which batch the cells belong to with shape `batch_size` :type y: `Optional[Tensor]` :param y: tensor of cell-types labels with shape (batch_size, n_labels) :type mode: `Optional[int]` :param mode: indicates which head/tail to use in the joint network :rtype: `Tuple[Tensor, Tensor]` :return: the reconstruction loss and the Kullback divergences

get_sample_rate(*x*, *batch_index*, *_, **__)

reconstruction_loss(*x*, *px_rate*, *px_r*, *px_dropout*, *mode*)

Return type `Tensor`

sample_from_posterior_l(*x*, *mode=None*, *deterministic=False*)

Sample the tensor of library sizes from the posterior

Parameters **x** (`Tensor`) – tensor of values with shape `(batch_size, n_input)`

or (batch_size, n_input_fish) depending on the mode :type mode: Optional[int] :param mode: head id to use in the encoder :type deterministic: bool :param deterministic: bool - whether to sample or not :rtype: Tensor :return: tensor of shape (batch_size, 1)

sample_from_posterior_z(*x*, *mode=None*, *deterministic=False*)
Sample tensor of latent values from the posterior

Parameters

- **x** (Tensor) – tensor of values with shape (batch_size, n_input)
- **mode** (Optional[int]) – head id to use in the encoder
- **deterministic** (bool) – bool - whether to sample or not

Return type Tensor

Returns tensor of shape (batch_size, n_latent)

sample_rate(*x*, *mode*, *batch_index*, *y=None*, *deterministic=False*, *decode_mode=None*)
Returns the tensor of scaled frequencies of expression

Parameters **x** (Tensor) – tensor of values with shape (batch_size, n_input)

or (batch_size, n_input_fish) depending on the mode :type y: Optional[Tensor] :param y: tensor of cell-types labels with shape (batch_size, n_labels) :type mode: int :param mode: int encode mode (which input head to use in the model) :type batch_index: Tensor :param batch_index: array that indicates which batch the cells belong to with shape batch_size :type deterministic: bool :param deterministic: bool - whether to sample or not :type decode_mode: Optional[int] :param decode_mode: int use to a decode mode different from encoding mode :rtype: Tensor :return: tensor of means of the scaled frequencies

sample_scale(*x*, *mode*, *batch_index*, *y=None*, *deterministic=False*, *decode_mode=None*)
Return the tensor of predicted frequencies of expression

Parameters **x** (Tensor) – tensor of values with shape (batch_size, n_input)

or (batch_size, n_input_fish) depending on the mode :type mode: int :param mode: int encode mode (which input head to use in the model) :type batch_index: Tensor :param batch_index: array that indicates which batch the cells belong to with shape batch_size :type y: Optional[Tensor] :param y: tensor of cell-types labels with shape (batch_size, n_labels) :type deterministic: bool :param deterministic: bool - whether to sample or not :type decode_mode: Optional[int] :param decode_mode: int use to a decode mode different from encoding mode :rtype: Tensor :return: tensor of predicted expression

scvi.models.log_likelihood module

File for computing log likelihood of the data

scvi.models.log_likelihood.**compute_elbo**(*vae*, *posterior*, ***kwargs*)
Computes the ELBO.

The ELBO is the reconstruction error + the KL divergences between the variational distributions and the priors. It differs from the marginal log likelihood. Specifically, it is a lower bound on the marginal log likelihood plus a term that is constant with respect to the variational distribution. It still gives good insights on the modeling of the data, and is fast to compute.

scvi.models.log_likelihood.**compute_marginal_log_likelihood_autozi**(*autozivae*, *posterior*, *n_samples_mc=100*)
Computes a biased estimator for log p(x), which is the marginal log likelihood.

Despite its bias, the estimator still converges to the real value of log p(x) when n_samples_mc (for Monte Carlo) goes to infinity (a fairly high value like 100 should be enough) Due to the Monte Carlo sampling, this method is not as computationally efficient as computing only the reconstruction loss

scvi.models.log_likelihood.**compute_marginal_log_likelihood_scvi**(*vae*, *posterior*, *n_samples_mc=100*)

Computes a biased estimator for log p(x), which is the marginal log likelihood.

Despite its bias, the estimator still converges to the real value of log p(x) when n_samples_mc (for Monte Carlo) goes to infinity (a fairly high value like 100 should be enough) Due to the Monte Carlo sampling, this method is not as computationally efficient as computing only the reconstruction loss

scvi.models.log_likelihood.**compute_reconstruction_error**(*vae*, *posterior*, ***kwargs*)

Computes log p(x/z), which is the reconstruction error.

Differs from the marginal log likelihood, but still gives good insights on the modeling of the data, and is fast to compute.

scvi.models.log_likelihood.**log_mixture_nb**(*x*, *mu_1*, *mu_2*, *theta_1*, *theta_2*, *pi*, *eps=1e-08*)

Note: All inputs should be torch Tensors log likelihood (scalar) of a minibatch according to a mixture nb model. pi is the probability to be in the first component.

For totalVI, the first component should be background.

Variables: mu1: mean of the first negative binomial component (has to be positive support) (shape: minibatch x genes) theta1: first inverse dispersion parameter (has to be positive support) (shape: minibatch x genes) mu2: mean of the second negative binomial (has to be positive support) (shape: minibatch x genes) theta2: second inverse dispersion parameter (has to be positive support) (shape: minibatch x genes)

If None, assume one shared inverse dispersion parameter.

eps: numerical stability constant

scvi.models.log_likelihood.**log_nb_positive**(*x*, *mu*, *theta*, *eps=1e-08*)

Note: All inputs should be torch Tensors log likelihood (scalar) of a minibatch according to a nb model.

Variables: mu: mean of the negative binomial (has to be positive support) (shape: minibatch x genes) theta: inverse dispersion parameter (has to be positive support) (shape: minibatch x genes) eps: numerical stability constant

scvi.models.log_likelihood.**log_zinb_positive**(*x*, *mu*, *theta*, *pi*, *eps=1e-08*)

Note: All inputs are torch Tensors log likelihood (scalar) of a minibatch according to a zinb model. Notes: We parametrize the bernoulli using the logits, hence the softplus functions appearing

Variables: mu: mean of the negative binomial (has to be positive support) (shape: minibatch x genes) theta: inverse dispersion parameter (has to be positive support) (shape: minibatch x genes) pi: logit of the dropout parameter (real support) (shape: minibatch x genes) eps: numerical stability constant

scvi.models.log_likelihood.**logsumexp**(*input*, *dim*, *keepdim=False*, *out=None*)

Returns the log of summed exponentials of each row of the `input` tensor in the given dimension `dim`. The computation is numerically stabilized.

For summation index j given by *dim* and other indices i, the result is

$$\text{logsumexp}(x)_i = \log \sum_j \exp(x_{ij})$$

If `keepdim` is `True`, the output tensor is of the same size as `input` except in the dimension(s) `dim` where it is of size 1. Otherwise, `dim` is squeezed (see `torch.squeeze()`), resulting in the output tensor having 1 (or `len(dim)`) fewer dimension(s).

Args: input (Tensor): the input tensor. dim (int or tuple of ints): the dimension or dimensions to reduce. keepdim (bool): whether the output tensor has `dim` retained or not. out (Tensor, optional): the output tensor.

Example::

```
>>> a = torch.randn(3, 3)
>>> torch.logsumexp(a, 1)
tensor([ 0.8442,  1.4322,  0.8711])
```

scvi.models.modules module

class scvi.models.modules.**Decoder**(*n_input*, *n_output*, *n_cat_list=None*, *n_layers=1*, *n_hidden=128*)

Bases: torch.nn.modules.module.Module

Decodes data from latent space of `n_input` dimensions to `n_output` dimensions using a fully-connected neural network of `n_hidden` layers. Output is the mean and variance of a multivariate Gaussian

Parameters

- **n_input** (int) – The dimensionality of the input (latent space)
- **n_output** (int) – The dimensionality of the output (data space)
- **n_cat_list** (Optional[Iterable[int]]) – A list containing the number of categories for each category of interest. Each category will be included using a one-hot encoding
- **n_layers** (int) – The number of fully-connected hidden layers
- **n_hidden** (int) – The number of nodes per hidden layer
- **dropout_rate** – Dropout rate to apply to each of the hidden layers

forward(*x*, **cat_list*)

The forward computation for a single sample.

1. Decodes the data from the latent space using the decoder network
2. Returns tensors for the mean and variance of a multivariate distribution

Parameters

- **x** (Tensor) – tensor with shape `(n_input,)`
- **cat_list** (int) – list of category membership(s) for this sample

Returns Mean and variance tensors of shape `(n_output,)`

Return type 2-tuple of `torch.Tensor`

class scvi.models.modules.**DecoderSCVI**(*n_input*, *n_output*, *n_cat_list=None*, *n_layers=1*, *n_hidden=128*)

Bases: torch.nn.modules.module.Module

Decodes data from latent space of `n_input` dimensions `n_output` dimensions using a fully-connected neural network of `n_hidden` layers.

Parameters

- **n_input** (int) – The dimensionality of the input (latent space)

- **n_output** (int) – The dimensionality of the output (data space)
- **n_cat_list** (Optional[Iterable[int]]) – A list containing the number of categories for each category of interest. Each category will be included using a one-hot encoding
- **n_layers** (int) – The number of fully-connected hidden layers
- **n_hidden** (int) – The number of nodes per hidden layer
- **dropout_rate** – Dropout rate to apply to each of the hidden layers

forward(*dispersion*, *z*, *library*, **cat_list*)

The forward computation for a single sample.

1. Decodes the data from the latent space using the decoder network
2. Returns parameters for the ZINB distribution of expression
3. If `dispersion != 'gene-cell'` then value for that param will be `None`

Parameters

- **dispersion** (str) – One of the following
  - `'gene'` - dispersion parameter of NB is constant per gene across cells
  - `'gene-batch'` - dispersion can differ between different batches
  - `'gene-label'` - dispersion can differ between different labels
  - `'gene-cell'` - dispersion can differ for every gene in every cell
- **z** (Tensor) – tensor with shape `(n_input,)`
- **library** (Tensor) – library size
- **cat_list** (int) – list of category membership(s) for this sample

Returns parameters for the ZINB distribution of expression

Return type 4-tuple of `torch.Tensor`

class scvi.models.modules.**DecoderTOTALVI**(*n_input*, *n_output_genes*, *n_output_proteins*, *n_cat_list=None*, *n_layers=1*, *n_hidden=256*, *dropout_rate=0*)

Bases: `torch.nn.modules.module.Module`

Decodes data from latent space of `n_input` dimensions `n_output` dimensions using a linear decoder

Parameters

- **n_input** (int) – The dimensionality of the input (latent space)
- **n_output_genes** (int) – The dimensionality of the output (gene space)
- **n_output_proteins** (int) – The dimensionality of the output (protein space)
- **n_cat_list** (Optional[Iterable[int]]) – A list containing the number of categories for each category of interest. Each category will be included using a one-hot encoding

forward(*z*, *library_gene*, **cat_list*)

The forward computation for a single sample.

1. Decodes the data from the latent space using the decoder network
2. Returns local parameters for the ZINB distribution for genes

3. Returns local parameters for the Mixture NB distribution for proteins

We use the dictionary *px_* to contain the parameters of the ZINB/NB for genes. The rate refers to the mean of the NB, dropout refers to Bernoulli mixing parameters. *scale* refers to the quanity upon which differential expression is performed. For genes, this can be viewed as the mean of the underlying gamma distribution.

We use the dictionary *py_* to contain the parameters of the Mixture NB distribution for proteins. *rate_fore* refers to foreground mean, while *rate_back* refers to background mean. *scale* refers to foreground mean adjusted for background probability and scaled to reside in simplex. *back_alpha* and *back_beta* are the posterior parameters for *rate_back*. *fore_scale* is the scaling factor that enforces *rate_fore* > *rate_back*.

Parameters

- **z** (Tensor) – tensor with shape (n_input,)
- **library_gene** (Tensor) – library size
- **cat_list** (int) – list of category membership(s) for this sample

Returns parameters for the ZINB distribution of expression

Return type 3-tuple (first 2-tuple dict, last torch.Tensor)

class scvi.models.modules.**Encoder**(*n_input*, *n_output*, *n_cat_list=None*, *n_layers=1*, *n_hidden=128*, *dropout_rate=0.1*)

Bases: torch.nn.modules.module.Module

Encodes data of n_input dimensions into a latent space of n_output dimensions using a fully-connected neural network of n_hidden layers.

Parameters

- **n_input** (int) – The dimensionality of the input (data space)
- **n_output** (int) – The dimensionality of the output (latent space)
- **n_cat_list** (Optional[Iterable[int]]) – A list containing the number of categories for each category of interest. Each category will be included using a one-hot encoding
- **n_layers** (int) – The number of fully-connected hidden layers
- **n_hidden** (int) – The number of nodes per hidden layer

Dropout_rate Dropout rate to apply to each of the hidden layers

forward(*x*, **cat_list*)

The forward computation for a single sample.

1. Encodes the data into latent space using the encoder network
2. Generates a mean \(q_m \) and variance \(q_v \) (clamped to \([-5, 5] \))
3. Samples a new value from an i.i.d. multivariate normal \(\sim N(q_m, \mathbf{I}q_v) \)

Parameters

- **x** (Tensor) – tensor with shape (n_input,)
- **cat_list** (int) – list of category membership(s) for this sample

Returns tensors of shape (n_latent,) for mean and var, and sample

Return type 3-tuple of torch.Tensor

class scvi.models.modules.**EncoderTOTALVI**(*n_input*, *n_output*, *n_cat_list=None*, *n_layers=2*, *n_hidden=256*, *dropout_rate=0.1*, *distribution='ln'*)

Bases: torch.nn.modules.module.Module

Encodes data of n_input dimensions into a latent space of n_output dimensions using a fully-connected neural network of n_hidden layers. :type n_input: int :param n_input: The dimensionality of the input (data space) :type n_output: int :param n_output: The dimensionality of the output (latent space) :type n_cat_list: Optional[Iterable[int]] :param n_cat_list: A list containing the number of categories

for each category of interest. Each category will be included using a one-hot encoding

Parameters

- **n_layers** (int) – The number of fully-connected hidden layers
- **n_hidden** (int) – The number of nodes per hidden layer

Dropout_rate Dropout rate to apply to each of the hidden layers

Distribution Distribution of the latent space, one of

- 'normal' - Normal distribution
- 'ln' - Logistic normal

forward(*data*, **cat_list*)

The forward computation for a single sample.

1. Encodes the data into latent space using the encoder network
2. Generates a mean \(q_m \) and variance \(q_v \)
3. Samples a new value from an i.i.d. latent distribution

The dictionary *latent* contains the samples of the latent variables, while *untran_latent* contains the untransformed versions of these latent variables. For example, the library size is log normally distributed, so *untran_latent["l"]* gives the normal sample that was later exponentiated to become *latent["l"]*. The logistic normal distribution is equivalent to applying softmax to a normal sample.

Parameters

- **data** (Tensor) – tensor with shape (n_input,)
- **cat_list** (int) – list of category membership(s) for this sample

Returns tensors of shape (n_latent,) for mean and var, and sample

Return type 6-tuple. First 4 of torch.Tensor, next 2 are *dict* of torch.Tensor

reparameterize_transformation(*mu*, *var*)

class scvi.models.modules.**FCLayers**(*n_in*, *n_out*, *n_cat_list=None*, *n_layers=1*, *n_hidden=128*, *dropout_rate=0.1*, *use_batch_norm=True*, *use_relu=True*, *bias=True*)

Bases: torch.nn.modules.module.Module

A helper class to build fully-connected layers for a neural network.

Parameters

- **n_in** (int) – The dimensionality of the input
- **n_out** (int) – The dimensionality of the output

- **n_cat_list** (Optional[Iterable[int]]) – A list containing, for each category of interest, the number of categories. Each category will be included using a one-hot encoding.
- **n_layers** (int) – The number of fully-connected hidden layers
- **n_hidden** (int) – The number of nodes per hidden layer
- **dropout_rate** (float) – Dropout rate to apply to each of the hidden layers
- **use_batch_norm** (bool) – Whether to have *BatchNorm* layers or not
- **use_relu** (bool) – Whether to have *ReLU* layers or not
- **bias** (bool) – Whether to learn bias in linear layers or not

forward(*x*, **cat_list*, *instance_id=0*)

Forward computation on x.

Parameters

- **x** (Tensor) – tensor of values with shape (n_in,)
- **cat_list** (int) – list of category membership(s) for this sample
- **instance_id** (int) – Use a specific conditional instance normalization (batchnorm)

Returns tensor of shape (n_out,)

Return type torch.Tensor

class scvi.models.modules.**LinearDecoderSCVI**(*n_input*, *n_output*, *n_cat_list=None*, *n_layers=1*, *n_hidden=128*)

Bases: torch.nn.modules.module.Module

forward(*dispersion*, *z*, *library*, **cat_list*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class scvi.models.modules.**MultiDecoder**(*n_input*, *n_output*, *n_hidden_conditioned=32*, *n_hidden_shared=128*, *n_layers_conditioned=1*, *n_layers_shared=1*, *n_cat_list=None*, *dropout_rate=0.2*)

Bases: torch.nn.modules.module.Module

forward(*z*, *dataset_id*, *library*, *dispersion*, **cat_list*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class scvi.models.modules.**MultiEncoder**(*n_heads*, *n_input_list*, *n_output*, *n_hidden=128*, *n_layers_individual=1*, *n_layers_shared=2*, *n_cat_list=None*, *dropout_rate=0.1*)

Bases: torch.nn.modules.module.Module

forward(*x*, *head_id*, **cat_list*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

scvi.models.modules.**reparameterize_gaussian**(*mu*, *var*)

scvi.models.scanvi module

class scvi.models.scanvi.**SCANVI**(*n_input*, *n_batch=0*, *n_labels=0*, *n_hidden=128*, *n_latent=10*, *n_layers=1*, *dropout_rate=0.1*, *dispersion='gene'*, *log_variational=True*, *reconstruction_loss='zinb'*, *y_prior=None*, *labels_groups=None*, *use_labels_groups=False*, *classifier_parameters={}*)

Bases: *scvi.models.vae.VAE*

A semi-supervised Variational auto-encoder model - inspired from M1 + M2 model, as described in (https://arxiv.org/pdf/1406.5298.pdf). SCANVI stands for single-cell annotation using variational inference.

Parameters

- **n_input** (int) – Number of input genes
- **n_batch** (int) – Number of batches
- **n_labels** (int) – Number of labels
- **n_hidden** (int) – Number of nodes per hidden layer
- **n_latent** (int) – Dimensionality of the latent space
- **n_layers** (int) – Number of hidden layers used for encoder and decoder NNs
- **dropout_rate** (float) – Dropout rate for neural networks
- **dispersion** (str) – One of the following
  - 'gene' - dispersion parameter of NB is constant per gene across cells
  - 'gene-batch' - dispersion can differ between different batches
  - 'gene-label' - dispersion can differ between different labels
  - 'gene-cell' - dispersion can differ for every gene in every cell
- **log_variational** (bool) – Log(data+1) prior to encoding for numerical stability. Not normalization.
- **reconstruction_loss** (str) – One of
  - 'nb' - Negative binomial distribution
  - 'zinb' - Zero-inflated negative binomial distribution

- **y_prior** – If None, initialized to uniform probability over cell types
- **labels_groups** (Optional[Sequence[int]]) – Label group designations
- **use_labels_groups** (bool) – Whether to use the label groups

Examples:

```
>>> gene_dataset = CortexDataset()
>>> scanvi = SCANVI(gene_dataset.nb_genes, n_batch=gene_dataset.n_batches *␣
↪False,
... n_labels=gene_dataset.n_labels)
```

```
>>> gene_dataset = SyntheticDataset(n_labels=3)
>>> scanvi = SCANVI(gene_dataset.nb_genes, n_batch=gene_dataset.n_batches *␣
↪False,
... n_labels=3, y_prior=torch.tensor([[0.1,0.5,0.4]]), labels_groups=[0,0,1])
```

classify(*x*)

forward(*x*, *local_l_mean*, *local_l_var*, *batch_index=None*, *y=None*)

Returns the reconstruction loss and the Kullback divergences

Parameters

- **x** – tensor of values with shape (batch_size, n_input)
- **local_l_mean** – tensor of means of the prior distribution of latent variable l with shape (batch_size, 1)
- **local_l_var** – tensor of variancess of the prior distribution of latent variable l with shape (batch_size, 1)
- **batch_index** – array that indicates which batch the cells belong to with shape batch_size
- **y** – tensor of cell-types labels with shape (batch_size, n_labels)

Returns the reconstruction loss and the Kullback divergences

Return type 2-tuple of torch.FloatTensor

get_latents(*x*, *y=None*)

returns the result of sample_from_posterior_z inside a list

Parameters

- **x** – tensor of values with shape (batch_size, n_input)
- **y** – tensor of cell-types labels with shape (batch_size, n_labels)

Returns one element list of tensor

Return type list of torch.Tensor

scvi.models.totalvi module

Main module.

class scvi.models.totalvi.**TOTALVI**(*n_input_genes*, *n_input_proteins*, *n_batch=0*, *n_labels=0*, *n_hidden=256*, *n_latent=20*, *n_layers=1*, *dropout_rate_decoder=0.2*, *dropout_rate_encoder=0.2*, *gene_dispersion='gene'*, *protein_dispersion='protein'*, *log_variational=True*, *reconstruction_loss_gene='nb'*, *latent_distribution='ln'*)

Bases: torch.nn.modules.module.Module

Latent variable model for CITE-seq data using auto-encoding Variational Bayes

Parameters

- **n_input_genes** (int) – Number of input genes
- **n_input_proteins** (int) – Number of input proteins
- **n_batch** (int) – Number of batches
- **n_labels** (int) – Number of labels
- **n_hidden** (int) – Number of nodes per hidden layer for the z encoder (protein+genes), genes library encoder, z->genes+proteins decoder
- **n_latent** (int) – Dimensionality of the latent space
- **n_layers** (int) – Number of hidden layers used for encoder and decoder NNs
- **dropout_rate** – Dropout rate for neural networks
- **genes_dispersion** – One of the following
  - 'gene' - genes_dispersion parameter of NB is constant per gene across cells
  - 'gene-batch' - genes_dispersion can differ between different batches
  - 'gene-label' - genes_dispersion can differ between different labels
- **protein_dispersion** (str) – One of the following
  - 'protein' - protein_dispersion parameter is constant per protein across cells
  - 'protein-batch' - protein_dispersion can differ between different batches NOT TESTED
  - 'protein-label' - protein_dispersion can differ between different labels NOT TESTED
- **log_variational** (bool) – Log(data+1) prior to encoding for numerical stability. Not normalization.
- **reconstruction_loss_genes** – One of
  - 'nb' - Negative binomial distribution
  - 'zinb' - Zero-inflated negative binomial distribution
- **latent_distribution** (str) – One of
  - 'normal' - Isotropic normal
  - 'ln' - Logistic normal with normal params N(0, 1)

Examples:

```
>>> dataset = Dataset10X(dataset_name="pbmc_10k_protein_v3", save_path=save_
↪path)
>>> totalvae = totalVI(gene_dataset.nb_genes, len(dataset.protein_names), use_
↪cuda=True )
```

forward (*x*, *y*, *local_l_mean_gene*, *local_l_var_gene*, *batch_index=None*, *label=None*)
Returns the reconstruction loss and the Kullback divergences

Parameters

- **x** (Tensor) – tensor of values with shape (batch_size, n_input_genes)
- **y** (Tensor) – tensor of values with shape (batch_size, n_input_proteins)
- **local_l_mean_gene** (Tensor) – tensor of means of the prior distribution of latent variable l with shape (batch_size, 1)
- **local_l_var_gene** (Tensor) – tensor of variancess of the prior distribution of latent variable l with shape (batch_size, 1)
- **batch_index** (Optional[Tensor]) – array that indicates which batch the cells belong to with shape batch_size
- **label** (Optional[Tensor]) – tensor of cell-types labels with shape (batch_size, n_labels)

Returns the reconstruction loss and the Kullback divergences

Return type 4-tuple of torch.FloatTensor

get_reconstruction_loss (*x*, *y*, *px_*, *py_*)

Return type Tuple[Tensor, Tensor]

get_sample_dispersion (*x*, *y*, *batch_index=None*, *label=None*, *n_samples=1*)
Returns the tensors of dispersions for genes and proteins

Parameters

- **x** (Tensor) – tensor of values with shape (batch_size, n_input_genes)
- **y** (Tensor) – tensor of values with shape (batch_size, n_input_proteins)
- **batch_index** (Optional[Tensor]) – array that indicates which batch the cells belong to with shape batch_size
- **label** (Optional[Tensor]) – tensor of cell-types labels with shape (batch_size, n_labels)
- **n_samples** (int) – number of samples

Returns tensor of means of the negative binomial distribution with shape (batch_size, n_input)

Return type 2-tuple of torch.Tensor

get_sample_rate (*x*, *y*, *batch_index=None*, *label=None*, *n_samples=1*)
Returns the tensor of negative binomial mean for genes

Parameters

- **x** (Tensor) – tensor of values with shape (batch_size, n_input_genes)
- **y** (Tensor) – tensor of values with shape (batch_size, n_input_proteins)

- **batch_index** (Optional[Tensor]) – array that indicates which batch the cells belong to with shape batch_size
- **label** (Optional[Tensor]) – tensor of cell-types labels with shape (batch_size, n_labels)
- **n_samples** (int) – number of samples

Returns tensor of means of the negative binomial distribution with shape (batch_size, n_input_genes)

Return type torch.Tensor

inference(*x*, *y*, *batch_index=None*, *label=None*, *n_samples=1*)

Internal helper function to compute necessary inference quantities

We use the dictionary *px_* to contain the parameters of the ZINB/NB for genes. The rate refers to the mean of the NB, dropout refers to Bernoulli mixing parameters. *scale* refers to the quanity upon which differential expression is performed. For genes, this can be viewed as the mean of the underlying gamma distribution.

We use the dictionary *py_* to contain the parameters of the Mixture NB distribution for proteins. *rate_fore* refers to foreground mean, while *rate_back* refers to background mean. *scale* refers to foreground mean adjusted for background probability and scaled to reside in simplex. *back_alpha* and *back_beta* are the posterior parameters for *rate_back*. *fore_scale* is the scaling factor that enforces *rate_fore* > *rate_back*.

px_["r"] and *py_["r"]* are the inverse dispersion parameters for genes and protein, respectively.

Return type Dict[str, Union[Tensor, Dict[str, Tensor]]]

sample_from_posterior_l(*x*, *y*, *batch_index=None*, *give_mean=True*)

samples the tensor of library size from the posterior #doesn't really sample, returns the tensor of the means of the posterior distribution

Parameters

- **x** (Tensor) – tensor of values with shape (batch_size, n_input_genes)
- **y** (Tensor) – tensor of values with shape (batch_size, n_input_proteins)

Returns tensor of shape (batch_size, 1)

Return type torch.Tensor

sample_from_posterior_z(*x*, *y*, *batch_index=None*, *give_mean=False*, *n_samples=5000*)

samples the tensor of latent values from the posterior #doesn't really sample, returns the means of the posterior distribution

Parameters

- **x** (Tensor) – tensor of values with shape (batch_size, n_input_genes)
- **y** (Tensor) – tensor of values with shape (batch_size, n_input_proteins)

Returns tensor of shape (batch_size, n_latent)

Return type torch.Tensor

scale_from_z(*x*, *y*, *fixed_batch*)

Returns tuple of gene and protein scales for a fixed seq batch

This function is the core of differential expression.

Return type Tensor

scvi.models.utils module

scvi.models.utils.**broadcast_labels**(*y*, **o*, *n_broadcast=-1*)
Utility for the semi-supervised setting If y is defined(labelled batch) then one-hot encode the labels (no broadcasting needed) If y is undefined (unlabelled batch) then generate all possible labels (and broadcast other arguments if not None)

scvi.models.utils.**enumerate_discrete**(*x*, *y_dim*)

scvi.models.utils.**iterate**(*obj*, *func*)

scvi.models.utils.**one_hot**(*index*, *n_cat*)

scvi.models.vae module

Main module.

class scvi.models.vae.**LDVAE**(*n_input*, *n_batch=0*, *n_labels=0*, *n_hidden=128*, *n_latent=10*, *n_layers=1*, *dropout_rate=0.1*, *dispersion='gene'*, *log_variational=True*, *reconstruction_loss='zinb'*)
Bases: *scvi.models.vae.VAE*

Linear-decoded Variational auto-encoder model.

This model uses a linear decoder, directly mapping the latent representation to gene expression levels. It still uses a deep neural network to encode the latent representation.

Compared to standard VAE, this model is less powerful, but can be used to inspect which genes contribute to variation in the dataset.

Parameters

- **n_input** (int) – Number of input genes
- **n_batch** (int) – Number of batches
- **n_labels** (int) – Number of labels
- **n_hidden** (int) – Number of nodes per hidden layer (for encoder)
- **n_latent** (int) – Dimensionality of the latent space
- **n_layers** (int) – Number of hidden layers used for encoder NNs
- **dropout_rate** (float) – Dropout rate for neural networks
- **dispersion** (str) – One of the following
  - 'gene' - dispersion parameter of NB is constant per gene across cells
  - 'gene-batch' - dispersion can differ between different batches
  - 'gene-label' - dispersion can differ between different labels
  - 'gene-cell' - dispersion can differ for every gene in every cell
- **log_variational** (bool) – Log(data+1) prior to encoding for numerical stability. Not normalization.
- **reconstruction_loss** (str) – One of
  - 'nb' - Negative binomial distribution
  - 'zinb' - Zero-inflated negative binomial distribution

get_loadings()
Extract per-gene weights (for each Z) in the linear decoder.

class scvi.models.vae.**VAE**(*n_input*, *n_batch=0*, *n_labels=0*, *n_hidden=128*, *n_latent=10*, *n_layers=1*, *dropout_rate=0.1*, *dispersion='gene'*, *log_variational=True*, *reconstruction_loss='zinb'*)
Bases: torch.nn.modules.module.Module

Variational auto-encoder model.

Parameters

- **n_input** (int) – Number of input genes
- **n_batch** (int) – Number of batches
- **n_labels** (int) – Number of labels
- **n_hidden** (int) – Number of nodes per hidden layer
- **n_latent** (int) – Dimensionality of the latent space
- **n_layers** (int) – Number of hidden layers used for encoder and decoder NNs
- **dropout_rate** (float) – Dropout rate for neural networks
- **dispersion** (str) – One of the following
  - 'gene' - dispersion parameter of NB is constant per gene across cells
  - 'gene-batch' - dispersion can differ between different batches
  - 'gene-label' - dispersion can differ between different labels
  - 'gene-cell' - dispersion can differ for every gene in every cell
- **log_variational** (bool) – Log(data+1) prior to encoding for numerical stability. Not normalization.
- **reconstruction_loss** (str) – One of
  - 'nb' - Negative binomial distribution
  - 'zinb' - Zero-inflated negative binomial distribution

Examples:

```
>>> gene_dataset = CortexDataset()
>>> vae = VAE(gene_dataset.nb_genes, n_batch=gene_dataset.n_batches * False,
... n_labels=gene_dataset.n_labels)
```

forward(*x*, *local_l_mean*, *local_l_var*, *batch_index=None*, *y=None*)
Returns the reconstruction loss and the Kullback divergences

Parameters

- **x** – tensor of values with shape (batch_size, n_input)
- **local_l_mean** – tensor of means of the prior distribution of latent variable l with shape (batch_size, 1)
- **local_l_var** – tensor of variancess of the prior distribution of latent variable l with shape (batch_size, 1)
- **batch_index** – array that indicates which batch the cells belong to with shape batch_size

- **y** – tensor of cell-types labels with shape (batch_size, n_labels)

Returns the reconstruction loss and the Kullback divergences

Return type 2-tuple of `torch.FloatTensor`

get_latents(*x*, *y=None*)
returns the result of `sample_from_posterior_z` inside a list

Parameters

- **x** – tensor of values with shape `(batch_size, n_input)`
- **y** – tensor of cell-types labels with shape `(batch_size, n_labels)`

Returns one element list of tensor

Return type list of `torch.Tensor`

get_reconstruction_loss(*x*, *px_rate*, *px_r*, *px_dropout*, ***kwargs*)

get_sample_rate(*x*, *batch_index=None*, *y=None*, *n_samples=1*)
Returns the tensor of means of the negative binomial distribution

Parameters

- **x** – tensor of values with shape `(batch_size, n_input)`
- **y** – tensor of cell-types labels with shape `(batch_size, n_labels)`
- **batch_index** – array that indicates which batch the cells belong to with shape `batch_size`
- **n_samples** – number of samples

Returns tensor of means of the negative binomial distribution with shape `(batch_size, n_input)`

Return type `torch.Tensor`

get_sample_scale(*x*, *batch_index=None*, *y=None*, *n_samples=1*)
Returns the tensor of predicted frequencies of expression

Parameters

- **x** – tensor of values with shape `(batch_size, n_input)`
- **batch_index** – array that indicates which batch the cells belong to with shape `batch_size`
- **y** – tensor of cell-types labels with shape `(batch_size, n_labels)`
- **n_samples** – number of samples

Returns tensor of predicted frequencies of expression with shape `(batch_size, n_input)`

Return type `torch.Tensor`

inference(*x*, *batch_index=None*, *y=None*, *n_samples=1*)

sample_from_posterior_l(*x*)
samples the tensor of library sizes from the posterior #doesn't really sample, returns the tensor of the means of the posterior distribution

Parameters

- **x** – tensor of values with shape `(batch_size, n_input)`

- **y** – tensor of cell-types labels with shape `(batch_size, n_labels)`

Returns tensor of shape `(batch_size, 1)`

Return type `torch.Tensor`

sample_from_posterior_z(*x*, *y=None*, *give_mean=False*)
samples the tensor of latent values from the posterior #doesn't really sample, returns the means of the posterior distribution

Parameters

- **x** – tensor of values with shape `(batch_size, n_input)`
- **y** – tensor of cell-types labels with shape `(batch_size, n_labels)`
- **give_mean** – is True when we want the mean of the posterior distribution rather than sampling

Returns tensor of shape `(batch_size, n_latent)`

Return type `torch.Tensor`

scale_from_z(*sample_batch*, *fixed_batch*)

scvi.models.vaec module

class scvi.models.vaec.**VAEC**(*n_input*, *n_batch*, *n_labels*, *n_hidden=128*, *n_latent=10*, *n_layers=1*, *dropout_rate=0.1*, *y_prior=None*, *dispersion='gene'*, *log_variational=True*, *reconstruction_loss='zinb'*)

Bases: *scvi.models.vae.VAE*

A semi-supervised Variational auto-encoder model - inspired from M2 model, as described in (https://arxiv.org/pdf/1406.5298.pdf)

Parameters

- **n_input** – Number of input genes
- **n_batch** – Number of batches
- **n_labels** – Number of labels
- **n_hidden** – Number of nodes per hidden layer
- **n_latent** – Dimensionality of the latent space
- **n_layers** – Number of hidden layers used for encoder and decoder NNs
- **dropout_rate** – Dropout rate for neural networks
- **dispersion** – One of the following
  - `'gene'` - dispersion parameter of NB is constant per gene across cells
  - `'gene-batch'` - dispersion can differ between different batches
  - `'gene-label'` - dispersion can differ between different labels
  - `'gene-cell'` - dispersion can differ for every gene in every cell
- **log_variational** – Log(data+1) prior to encoding for numerical stability. Not normalization.
- **reconstruction_loss** – One of
  - `'nb'` - Negative binomial distribution

- `'zinb'` - Zero-inflated negative binomial distribution

- **y_prior** – If None, initialized to uniform probability over cell types

Examples:

```
>>> gene_dataset = CortexDataset()
>>> vaec = VAEC(gene_dataset.nb_genes, n_batch=gene_dataset.n_batches * False,
... n_labels=gene_dataset.n_labels)
```

```
>>> gene_dataset = SyntheticDataset(n_labels=3)
>>> vaec = VAEC(gene_dataset.nb_genes, n_batch=gene_dataset.n_batches * False,
... n_labels=3, y_prior=torch.tensor([[0.1,0.5,0.4]]))
```

classify(*x*)

forward(*x*, *local_l_mean*, *local_l_var*, *batch_index=None*, *y=None*)

Returns the reconstruction loss and the Kullback divergences

Parameters

- **x** – tensor of values with shape (batch_size, n_input)
- **local_l_mean** – tensor of means of the prior distribution of latent variable l with shape (batch_size, 1)
- **local_l_var** – tensor of variancess of the prior distribution of latent variable l with shape (batch_size, 1)
- **batch_index** – array that indicates which batch the cells belong to with shape `batch_size`
- **y** – tensor of cell-types labels with shape (batch_size, n_labels)

Returns the reconstruction loss and the Kullback divergences

Return type 2-tuple of `torch.FloatTensor`

Module contents

class scvi.models.**SCANVI**(*n_input*, *n_batch=0*, *n_labels=0*, *n_hidden=128*, *n_latent=10*, *n_layers=1*, *dropout_rate=0.1*, *dispersion='gene'*, *log_variational=True*, *reconstruction_loss='zinb'*, *y_prior=None*, *labels_groups=None*, *use_labels_groups=False*, *classifier_parameters={}*)

Bases: *scvi.models.vae.VAE*

A semi-supervised Variational auto-encoder model - inspired from M1 + M2 model, as described in (https://arxiv.org/pdf/1406.5298.pdf). SCANVI stands for single-cell annotation using variational inference.

Parameters

- **n_input** (int) – Number of input genes
- **n_batch** (int) – Number of batches
- **n_labels** (int) – Number of labels
- **n_hidden** (int) – Number of nodes per hidden layer
- **n_latent** (int) – Dimensionality of the latent space
- **n_layers** (int) – Number of hidden layers used for encoder and decoder NNs

- **dropout_rate** (float) – Dropout rate for neural networks
- **dispersion** (str) – One of the following
  - 'gene' - dispersion parameter of NB is constant per gene across cells
  - 'gene-batch' - dispersion can differ between different batches
  - 'gene-label' - dispersion can differ between different labels
  - 'gene-cell' - dispersion can differ for every gene in every cell
- **log_variational** (bool) – Log(data+1) prior to encoding for numerical stability. Not normalization.
- **reconstruction_loss** (str) – One of
  - 'nb' - Negative binomial distribution
  - 'zinb' - Zero-inflated negative binomial distribution
- **y_prior** – If None, initialized to uniform probability over cell types
- **labels_groups** (Optional[Sequence[int]]) – Label group designations
- **use_labels_groups** (bool) – Whether to use the label groups

Examples:

```
>>> gene_dataset = CortexDataset()
>>> scanvi = SCANVI(gene_dataset.nb_genes, n_batch=gene_dataset.n_batches *␣
↪False,
... n_labels=gene_dataset.n_labels)
```

```
>>> gene_dataset = SyntheticDataset(n_labels=3)
>>> scanvi = SCANVI(gene_dataset.nb_genes, n_batch=gene_dataset.n_batches *␣
↪False,
... n_labels=3, y_prior=torch.tensor([[0.1,0.5,0.4]]), labels_groups=[0,0,1])
```

classify(*x*)

forward(*x*, *local_l_mean*, *local_l_var*, *batch_index=None*, *y=None*)

Returns the reconstruction loss and the Kullback divergences

Parameters

- **x** – tensor of values with shape (batch_size, n_input)
- **local_l_mean** – tensor of means of the prior distribution of latent variable l with shape (batch_size, 1)
- **local_l_var** – tensor of variancess of the prior distribution of latent variable l with shape (batch_size, 1)
- **batch_index** – array that indicates which batch the cells belong to with shape batch_size
- **y** – tensor of cell-types labels with shape (batch_size, n_labels)

Returns the reconstruction loss and the Kullback divergences

Return type 2-tuple of torch.FloatTensor

get_latents(*x*, *y=None*)

returns the result of sample_from_posterior_z inside a list

Parameters

- **x** – tensor of values with shape `(batch_size, n_input)`
- **y** – tensor of cell-types labels with shape `(batch_size, n_labels)`

Returns one element list of tensor

Return type list of `torch.Tensor`

class `scvi.models.`**VAEC**(*n_input*, *n_batch*, *n_labels*, *n_hidden=128*, *n_latent=10*, *n_layers=1*, *dropout_rate=0.1*, *y_prior=None*, *dispersion='gene'*, *log_variational=True*, *reconstruction_loss='zinb'*)

Bases: *scvi.models.vae.VAE*

A semi-supervised Variational auto-encoder model - inspired from M2 model, as described in (https://arxiv.org/pdf/1406.5298.pdf)

Parameters

- **n_input** – Number of input genes
- **n_batch** – Number of batches
- **n_labels** – Number of labels
- **n_hidden** – Number of nodes per hidden layer
- **n_latent** – Dimensionality of the latent space
- **n_layers** – Number of hidden layers used for encoder and decoder NNs
- **dropout_rate** – Dropout rate for neural networks
- **dispersion** – One of the following
  - `'gene'` - dispersion parameter of NB is constant per gene across cells
  - `'gene-batch'` - dispersion can differ between different batches
  - `'gene-label'` - dispersion can differ between different labels
  - `'gene-cell'` - dispersion can differ for every gene in every cell
- **log_variational** – Log(data+1) prior to encoding for numerical stability. Not normalization.
- **reconstruction_loss** – One of
  - `'nb'` - Negative binomial distribution
  - `'zinb'` - Zero-inflated negative binomial distribution
- **y_prior** – If None, initialized to uniform probability over cell types

Examples:

```
>>> gene_dataset = CortexDataset()
>>> vaec = VAEC(gene_dataset.nb_genes, n_batch=gene_dataset.n_batches * False,
... n_labels=gene_dataset.n_labels)
```

```
>>> gene_dataset = SyntheticDataset(n_labels=3)
>>> vaec = VAEC(gene_dataset.nb_genes, n_batch=gene_dataset.n_batches * False,
... n_labels=3, y_prior=torch.tensor([[0.1,0.5,0.4]]))
```

classify(*x*)

forward(*x*, *local_l_mean*, *local_l_var*, *batch_index=None*, *y=None*)
Returns the reconstruction loss and the Kullback divergences

Parameters

- **x** – tensor of values with shape (batch_size, n_input)
- **local_l_mean** – tensor of means of the prior distribution of latent variable l with shape (batch_size, 1)
- **local_l_var** – tensor of variancess of the prior distribution of latent variable l with shape (batch_size, 1)
- **batch_index** – array that indicates which batch the cells belong to with shape `batch_size`
- **y** – tensor of cell-types labels with shape (batch_size, n_labels)

Returns the reconstruction loss and the Kullback divergences

Return type 2-tuple of `torch.FloatTensor`

class scvi.models.**VAE**(*n_input*, *n_batch=0*, *n_labels=0*, *n_hidden=128*, *n_latent=10*, *n_layers=1*, *dropout_rate=0.1*, *dispersion='gene'*, *log_variational=True*, *reconstruction_loss='zinb'*)
Bases: `torch.nn.modules.module.Module`

Variational auto-encoder model.

Parameters

- **n_input** (`int`) – Number of input genes
- **n_batch** (`int`) – Number of batches
- **n_labels** (`int`) – Number of labels
- **n_hidden** (`int`) – Number of nodes per hidden layer
- **n_latent** (`int`) – Dimensionality of the latent space
- **n_layers** (`int`) – Number of hidden layers used for encoder and decoder NNs
- **dropout_rate** (`float`) – Dropout rate for neural networks
- **dispersion** (`str`) – One of the following
  - `'gene'` - dispersion parameter of NB is constant per gene across cells
  - `'gene-batch'` - dispersion can differ between different batches
  - `'gene-label'` - dispersion can differ between different labels
  - `'gene-cell'` - dispersion can differ for every gene in every cell
- **log_variational** (`bool`) – Log(data+1) prior to encoding for numerical stability. Not normalization.
- **reconstruction_loss** (`str`) – One of
  - `'nb'` - Negative binomial distribution
  - `'zinb'` - Zero-inflated negative binomial distribution

Examples:

```
>>> gene_dataset = CortexDataset()
>>> vae = VAE(gene_dataset.nb_genes, n_batch=gene_dataset.n_batches * False,
... n_labels=gene_dataset.n_labels)
```

forward(*x*, *local_l_mean*, *local_l_var*, *batch_index=None*, *y=None*)
Returns the reconstruction loss and the Kullback divergences

Parameters

- **x** – tensor of values with shape (batch_size, n_input)
- **local_l_mean** – tensor of means of the prior distribution of latent variable l with shape (batch_size, 1)
- **local_l_var** – tensor of variancess of the prior distribution of latent variable l with shape (batch_size, 1)
- **batch_index** – array that indicates which batch the cells belong to with shape `batch_size`
- **y** – tensor of cell-types labels with shape (batch_size, n_labels)

Returns the reconstruction loss and the Kullback divergences

Return type 2-tuple of `torch.FloatTensor`

get_latents(*x*, *y=None*)
returns the result of `sample_from_posterior_z` inside a list

Parameters

- **x** – tensor of values with shape `(batch_size, n_input)`
- **y** – tensor of cell-types labels with shape `(batch_size, n_labels)`

Returns one element list of tensor

Return type list of `torch.Tensor`

get_reconstruction_loss(*x*, *px_rate*, *px_r*, *px_dropout*, ***kwargs*)

get_sample_rate(*x*, *batch_index=None*, *y=None*, *n_samples=1*)
Returns the tensor of means of the negative binomial distribution

Parameters

- **x** – tensor of values with shape `(batch_size, n_input)`
- **y** – tensor of cell-types labels with shape `(batch_size, n_labels)`
- **batch_index** – array that indicates which batch the cells belong to with shape `batch_size`
- **n_samples** – number of samples

Returns tensor of means of the negative binomial distribution with shape `(batch_size, n_input)`

Return type `torch.Tensor`

get_sample_scale(*x*, *batch_index=None*, *y=None*, *n_samples=1*)
Returns the tensor of predicted frequencies of expression

Parameters

- **x** – tensor of values with shape `(batch_size, n_input)`

- **batch_index** – array that indicates which batch the cells belong to with shape `batch_size`
- **y** – tensor of cell-types labels with shape `(batch_size, n_labels)`
- **n_samples** – number of samples

Returns tensor of predicted frequencies of expression with shape `(batch_size, n_input)`

Return type `torch.Tensor`

inference(*x*, *batch_index=None*, *y=None*, *n_samples=1*)

sample_from_posterior_l(*x*)
samples the tensor of library sizes from the posterior #doesn't really sample, returns the tensor of the means of the posterior distribution

Parameters

- **x** – tensor of values with shape `(batch_size, n_input)`
- **y** – tensor of cell-types labels with shape `(batch_size, n_labels)`

Returns tensor of shape `(batch_size, 1)`

Return type `torch.Tensor`

sample_from_posterior_z(*x*, *y=None*, *give_mean=False*)
samples the tensor of latent values from the posterior #doesn't really sample, returns the means of the posterior distribution

Parameters

- **x** – tensor of values with shape `(batch_size, n_input)`
- **y** – tensor of cell-types labels with shape `(batch_size, n_labels)`
- **give_mean** – is True when we want the mean of the posterior distribution rather than sampling

Returns tensor of shape `(batch_size, n_latent)`

Return type `torch.Tensor`

scale_from_z(*sample_batch*, *fixed_batch*)

class scvi.models.**LDVAE**(*n_input*, *n_batch=0*, *n_labels=0*, *n_hidden=128*, *n_latent=10*, *n_layers=1*, *dropout_rate=0.1*, *dispersion='gene'*, *log_variational=True*, *reconstruction_loss='zinb'*)

Bases: *scvi.models.vae.VAE*

Linear-decoded Variational auto-encoder model.

This model uses a linear decoder, directly mapping the latent representation to gene expression levels. It still uses a deep neural network to encode the latent representation.

Compared to standard VAE, this model is less powerful, but can be used to inspect which genes contribute to variation in the dataset.

Parameters

- **n_input** (`int`) – Number of input genes
- **n_batch** (`int`) – Number of batches
- **n_labels** (`int`) – Number of labels

- **n_hidden** (int) – Number of nodes per hidden layer (for encoder)
- **n_latent** (int) – Dimensionality of the latent space
- **n_layers** (int) – Number of hidden layers used for encoder NNs
- **dropout_rate** (float) – Dropout rate for neural networks
- **dispersion** (str) – One of the following
  - 'gene' - dispersion parameter of NB is constant per gene across cells
  - 'gene-batch' - dispersion can differ between different batches
  - 'gene-label' - dispersion can differ between different labels
  - 'gene-cell' - dispersion can differ for every gene in every cell
- **log_variational** (bool) – Log(data+1) prior to encoding for numerical stability. Not normalization.
- **reconstruction_loss** (str) – One of
  - 'nb' - Negative binomial distribution
  - 'zinb' - Zero-inflated negative binomial distribution

get_loadings()
Extract per-gene weights (for each Z) in the linear decoder.

class scvi.models.**JVAE**(*dim_input_list*, *total_genes*, *indices_mappings*, *reconstruction_losses*, *model_library_bools*, *n_latent=10*, *n_layers_encoder_individual=1*, *n_layers_encoder_shared=1*, *dim_hidden_encoder=128*, *n_layers_decoder_individual=0*, *n_layers_decoder_shared=0*, *dim_hidden_decoder_individual=32*, *dim_hidden_decoder_shared=128*, *dropout_rate_encoder=0.1*, *dropout_rate_decoder=0.3*, *n_batch=0*, *n_labels=0*, *dispersion='gene-batch'*, *log_variational=True*)
Bases: torch.nn.modules.module.Module

Joint Variational auto-encoder

Implementation of gimVI: *A joint model of unpaired data from scRNA-seq and spatial transcriptomics for imputing missing gene expression measurements* https://arxiv.org/abs/1905.02269

decode(*z*, *mode*, *library*, *batch_index=None*, *y=None*)

Return type Tuple[Tensor, Tensor, Tensor, Tensor]

encode(*x*, *mode*)

Return type Tuple[Tensor, Tensor, Tensor, Optional[Tensor], Optional[Tensor], Tensor]

forward(*x*, *local_l_mean*, *local_l_var*, *batch_index=None*, *y=None*, *mode=None*)
Return the reconstruction loss and the Kullback divergences

Parameters **x** (Tensor) – tensor of values with shape (batch_size, n_input)

or (batch_size, n_input_fish) depending on the mode :type local_l_mean: Tensor :param local_l_mean: tensor of means of the prior distribution of latent variable l with shape (batch_size, 1) :type local_l_var: Tensor :param local_l_var: tensor of variances of the prior distribution of latent variable l with shape (batch_size, 1) :type batch_index: Optional[Tensor] :param batch_index: array that indicates which batch the cells belong to with shape batch_size :type y: Optional[Tensor] :param y: tensor of cell-types labels with shape (batch_size, n_labels) :type mode: Optional[int] :param mode: indicates which head/tail to use in the joint network :rtype: Tuple[Tensor, Tensor] :return: the reconstruction loss and the Kullback divergences

get_sample_rate(*x*, *batch_index*, *_, **__)

reconstruction_loss(*x*, *px_rate*, *px_r*, *px_dropout*, *mode*)

Return type Tensor

sample_from_posterior_l(*x*, *mode=None*, *deterministic=False*)
Sample the tensor of library sizes from the posterior

Parameters **x** (Tensor) – tensor of values with shape (batch_size, n_input)

or (batch_size, n_input_fish) depending on the mode :type mode: Optional[int] :param mode: head id to use in the encoder :type deterministic: bool :param deterministic: bool - whether to sample or not :rtype: Tensor :return: tensor of shape (batch_size, 1)

sample_from_posterior_z(*x*, *mode=None*, *deterministic=False*)
Sample tensor of latent values from the posterior

Parameters

- **x** (Tensor) – tensor of values with shape (batch_size, n_input)
- **mode** (Optional[int]) – head id to use in the encoder
- **deterministic** (bool) – bool - whether to sample or not

Return type Tensor

Returns tensor of shape (batch_size, n_latent)

sample_rate(*x*, *mode*, *batch_index*, *y=None*, *deterministic=False*, *decode_mode=None*)
Returns the tensor of scaled frequencies of expression

Parameters **x** (Tensor) – tensor of values with shape (batch_size, n_input)

or (batch_size, n_input_fish) depending on the mode :type y: Optional[Tensor] :param y: tensor of cell-types labels with shape (batch_size, n_labels) :type mode: int :param mode: int encode mode (which input head to use in the model) :type batch_index: Tensor :param batch_index: array that indicates which batch the cells belong to with shape batch_size :type deterministic: bool :param deterministic: bool - whether to sample or not :type decode_mode: Optional[int] :param decode_mode: int use to a decode mode different from encoding mode :rtype: Tensor :return: tensor of means of the scaled frequencies

sample_scale(*x*, *mode*, *batch_index*, *y=None*, *deterministic=False*, *decode_mode=None*)
Return the tensor of predicted frequencies of expression

Parameters **x** (Tensor) – tensor of values with shape (batch_size, n_input)

or (batch_size, n_input_fish) depending on the mode :type mode: int :param mode: int encode mode (which input head to use in the model) :type batch_index: Tensor :param batch_index: array that indicates which batch the cells belong to with shape batch_size :type y: Optional[Tensor] :param y: tensor of cell-types labels with shape (batch_size, n_labels) :type deterministic: bool :param deterministic: bool - whether to sample or not :type decode_mode: Optional[int] :param decode_mode: int use to a decode mode different from encoding mode :rtype: Tensor :return: tensor of predicted expression

class scvi.models.**Classifier**(*n_input*, *n_hidden=128*, *n_labels=5*, *n_layers=1*, *dropout_rate=0.1*, *logits=False*)
Bases: torch.nn.modules.module.Module

forward(*x*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

class scvi.models.**AutoZIVAE**(*n_input*, *alpha_prior=0.5*, *beta_prior=0.5*, *minimal_dropout=0.01*, *zero_inflation='gene'*, ***args*)

Bases: *scvi.models.vae.VAE*

compute_global_kl_divergence()

Return type Tensor

cuda(*device=None*)

Moves all model parameters and also fixed prior alpha and beta values, when relevant, to the GPU.

Parameters **device** (Optional[str]) – string denoting the GPU device on which parameters and prior distribution values are copied.

Return type Module

forward(*x*, *local_l_mean*, *local_l_var*, *batch_index=None*, *y=None*)

Returns the reconstruction loss and the Kullback divergences

Parameters

- **x** (Tensor) – tensor of values with shape (batch_size, n_input)
- **local_l_mean** (Tensor) – tensor of means of the prior distribution of latent variable l with shape (batch_size, 1)
- **local_l_var** (Tensor) – tensor of variancess of the prior distribution of latent variable l with shape (batch_size, 1)
- **batch_index** (Optional[Tensor]) – array that indicates which batch the cells belong to with shape batch_size
- **y** (Optional[Tensor]) – tensor of cell-types labels with shape (batch_size, n_labels)

Returns the reconstruction loss and the Kullback divergences

Return type 2-tuple of torch.FloatTensor

get_alphas_betas(*as_numpy=True*)

Return type Dict[str, Union[Tensor, ndarray]]

get_reconstruction_loss(*x*, *px_rate*, *px_r*, *px_dropout*, *bernoulli_params*, *eps_log=1e-08*, ***kwargs*)

Return type Tensor

inference(*x*, *batch_index=None*, *y=None*, *n_samples=1*, *eps_log=1e-08*)

Return type Dict[str, Tensor]

rescale_dropout(*px_dropout*, *eps_log=1e-08*)

Return type Tensor

reshape_bernoulli(*bernoulli_params*, *batch_index=None*, *y=None*)

Return type Tensor

sample_bernoulli_params(*batch_index=None*, *y=None*, *n_samples=1*)

Return type Tensor

sample_from_beta_distribution(*alpha*, *beta*, *eps_gamma=1e-30*, *eps_sample=1e-07*)

Return type Tensor

class scvi.models.**TOTALVI**(*n_input_genes*, *n_input_proteins*, *n_batch=0*, *n_labels=0*, *n_hidden=256*, *n_latent=20*, *n_layers=1*, *dropout_rate_decoder=0.2*, *dropout_rate_encoder=0.2*, *gene_dispersion='gene'*, *protein_dispersion='protein'*, *log_variational=True*, *reconstruction_loss_gene='nb'*, *latent_distribution='ln'*)

Bases: torch.nn.modules.module.Module

Latent variable model for CITE-seq data using auto-encoding Variational Bayes

Parameters

- **n_input_genes** (int) – Number of input genes
- **n_input_proteins** (int) – Number of input proteins
- **n_batch** (int) – Number of batches
- **n_labels** (int) – Number of labels
- **n_hidden** (int) – Number of nodes per hidden layer for the z encoder (protein+genes), genes library encoder, z->genes+proteins decoder
- **n_latent** (int) – Dimensionality of the latent space
- **n_layers** (int) – Number of hidden layers used for encoder and decoder NNs
- **dropout_rate** – Dropout rate for neural networks
- **genes_dispersion** – One of the following
  - 'gene' - genes_dispersion parameter of NB is constant per gene across cells
  - 'gene-batch' - genes_dispersion can differ between different batches
  - 'gene-label' - genes_dispersion can differ between different labels
- **protein_dispersion** (str) – One of the following
  - 'protein' - protein_dispersion parameter is constant per protein across cells
  - 'protein-batch' - protein_dispersion can differ between different batches NOT TESTED
  - 'protein-label' - protein_dispersion can differ between different labels NOT TESTED
- **log_variational** (bool) – Log(data+1) prior to encoding for numerical stability. Not normalization.
- **reconstruction_loss_genes** – One of
  - 'nb' - Negative binomial distribution
  - 'zinb' - Zero-inflated negative binomial distribution
- **latent_distribution** (str) – One of
  - 'normal' - Isotropic normal
  - 'ln' - Logistic normal with normal params N(0, 1)

Examples:

```
>>> dataset = Dataset10X(dataset_name="pbmc_10k_protein_v3", save_path=save_
↪path)
>>> totalvae = totalVI(gene_dataset.nb_genes, len(dataset.protein_names), use_
↪cuda=True )
```

forward (*x*, *y*, *local_l_mean_gene*, *local_l_var_gene*, *batch_index=None*, *label=None*)
Returns the reconstruction loss and the Kullback divergences

Parameters

- **x** (Tensor) – tensor of values with shape (batch_size, n_input_genes)
- **y** (Tensor) – tensor of values with shape (batch_size, n_input_proteins)
- **local_l_mean_gene** (Tensor) – tensor of means of the prior distribution of latent variable l with shape (batch_size, 1)
- **local_l_var_gene** (Tensor) – tensor of variancess of the prior distribution of latent variable l with shape (batch_size, 1)
- **batch_index** (Optional[Tensor]) – array that indicates which batch the cells belong to with shape batch_size
- **label** (Optional[Tensor]) – tensor of cell-types labels with shape (batch_size, n_labels)

Returns the reconstruction loss and the Kullback divergences

Return type 4-tuple of torch.FloatTensor

get_reconstruction_loss (*x*, *y*, *px_*, *py_*)

Return type Tuple[Tensor, Tensor]

get_sample_dispersion (*x*, *y*, *batch_index=None*, *label=None*, *n_samples=1*)
Returns the tensors of dispersions for genes and proteins

Parameters

- **x** (Tensor) – tensor of values with shape (batch_size, n_input_genes)
- **y** (Tensor) – tensor of values with shape (batch_size, n_input_proteins)
- **batch_index** (Optional[Tensor]) – array that indicates which batch the cells belong to with shape batch_size
- **label** (Optional[Tensor]) – tensor of cell-types labels with shape (batch_size, n_labels)
- **n_samples** (int) – number of samples

Returns tensor of means of the negative binomial distribution with shape (batch_size, n_input)

Return type 2-tuple of torch.Tensor

get_sample_rate (*x*, *y*, *batch_index=None*, *label=None*, *n_samples=1*)
Returns the tensor of negative binomial mean for genes

Parameters

- **x** (Tensor) – tensor of values with shape (batch_size, n_input_genes)
- **y** (Tensor) – tensor of values with shape (batch_size, n_input_proteins)

- **batch_index** (Optional[Tensor]) – array that indicates which batch the cells belong to with shape batch_size
- **label** (Optional[Tensor]) – tensor of cell-types labels with shape (batch_size, n_labels)
- **n_samples** (int) – number of samples

Returns tensor of means of the negative binomial distribution with shape (batch_size, n_input_genes)

Return type torch.Tensor

inference(*x*, *y*, *batch_index=None*, *label=None*, *n_samples=1*)

Internal helper function to compute necessary inference quantities

We use the dictionary *px_* to contain the parameters of the ZINB/NB for genes. The rate refers to the mean of the NB, dropout refers to Bernoulli mixing parameters. *scale* refers to the quanity upon which differential expression is performed. For genes, this can be viewed as the mean of the underlying gamma distribution.

We use the dictionary *py_* to contain the parameters of the Mixture NB distribution for proteins. *rate_fore* refers to foreground mean, while *rate_back* refers to background mean. *scale* refers to foreground mean adjusted for background probability and scaled to reside in simplex. *back_alpha* and *back_beta* are the posterior parameters for *rate_back*. *fore_scale* is the scaling factor that enforces *rate_fore* > *rate_back*.

px_["r"] and *py_["r"]* are the inverse dispersion parameters for genes and protein, respectively.

Return type Dict[str, Union[Tensor, Dict[str, Tensor]]]

sample_from_posterior_l(*x*, *y*, *batch_index=None*, *give_mean=True*)

samples the tensor of library size from the posterior #doesn't really sample, returns the tensor of the means of the posterior distribution

Parameters

- **x** (Tensor) – tensor of values with shape (batch_size, n_input_genes)
- **y** (Tensor) – tensor of values with shape (batch_size, n_input_proteins)

Returns tensor of shape (batch_size, 1)

Return type torch.Tensor

sample_from_posterior_z(*x*, *y*, *batch_index=None*, *give_mean=False*, *n_samples=5000*)

samples the tensor of latent values from the posterior #doesn't really sample, returns the means of the posterior distribution

Parameters

- **x** (Tensor) – tensor of values with shape (batch_size, n_input_genes)
- **y** (Tensor) – tensor of values with shape (batch_size, n_input_proteins)

Returns tensor of shape (batch_size, n_latent)

Return type torch.Tensor

scale_from_z(*x*, *y*, *fixed_batch*)

Returns tuple of gene and protein scales for a fixed seq batch

This function is the core of differential expression.

Return type Tensor

5.1.2 Module contents

Top-level package for scVI-dev.

scvi.**set_verbosity**(*level*)
Sets logging configuration for scvi based on chosen level of verbosity.

Sets "scvi" logging level to *level* If "scvi" logger has no StreamHandler, add one. Else, set its level to *level*.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

G

I

J

L

M

O

R

S

T

V