discern-reconstruction

Release 0.1.1

Fabian Hausmann <fabian.hausmann@zmnh.uni-hamburg.de>,Ca

Sep 22, 2023

CONTENTS:

CHAPTER

ONE

DISCERN

DISCERN is a deep learning approach to reconstruction expression of single-cell RNAseq data sets using a high-quality reference. Please also look at our manuscript DISCERN: deep single-cell expression reconstruction for improved cell clustering and cell subtype and state detection.

1.1 Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes. The full documentation contains further information and tutorials to get started.

1.1.1 Prerequisites

We use poetry for dependency management. You can get poetry by

```
pip install poetry
```

or (the officially recommended way)

```
curl -sSL https://raw.githubusercontent.com/python-poetry/poetry/master/get-poetry.py |␣
↪python
```

1.1.2 Installing

To get discern you can clone the repository by

```
git clone https://github.com/imsb-uke/discern.git
```

poetry can be used to install all further dependencies in a virtual environment.

```
cd discern
poetry install --no-dev
```

To finally run discern you can also directly use poetry with

```
poetry run commands
```

or spawn a new shell in the virtual environment

```
poetry shell
```

For all further examples, the first approach is used.

1.1.3 Using discern

You can use the main function of discern for most use cases. Usually, you have to preprocess your data by:

```
poetry run discern process <parameters.json>
```

An example parameters.json is provided together with an hyperparameter_search.json for hyperparameter optimization using ray[tune]. The training can be done with

```
poetry run discern train <parameters.json>
```

Hyperparameter optimization needs a ray server with can be started with

```
poetry run ray start --head --port 57780 --redis-password='password'
```

and can be started with

```
poetry run discern optimize <parameters.json>
```

For projection 2 different modes are available: Eval mode, which is a more general approach and can save a lot of files:

```
poetry run discern project --all_batches <parameters.json>
```

Or projection mode which offers a more fine grained controll to which is projected.

```
poetry run discern project --metadata="metadatacolumn:value" --metadata="metadatacolumn:
↪" <parameters.json>
```

which creates to files, one is projected to the average batch calculated by a `metadatacolumn` and a contained `value`. The second file is projected to the the average for each value in “metadatacolumn”; individually.

DISCERN also supports online training. You can add new batches to your dataset after the usual `train` with:

```
poetry run discern onlinetraining --freeze --filename=<new_not_preprocessed_batch[es].
↪h5ad> <parameters.json>
```

The data gets automatically preprocessed and added to the dataset. You can run `project` afterwards as usual (without the `--filename` flag). `--freeze` is important to freeze non-conditional layers in training.

1.2 Authors

- Can Ergen
- Pierre Machart
- Fabian Hausmann

1.3 Citation

If you use *discern* in your work, please cite the publication as follows:

> **DISCERN: deep single-cell expression reconstruction for improved cell clustering and cell subtype and state detection**
>
> Fabian Hausmann, Can Ergen, Robin Khatri, Mohamed Marouf, Sonja Hänzelmann, Nicola Gagliani, Samuel Huber, Pierre Machart & Stefan Bonn
>
> *Genome Biology* 2023 doi: 10.1186/s13059-023-03049-x.

CHAPTER
TWO

TUTORIALS

2.1 Small datasets and two batches

A very simple use case with DISCERN and two batches can be found in this Tutorial. It is recommended to start with this tutorial, when using DISCERN the first time. An interactive version can be found in Tutorial.ipynb.

2.2 Multiple-batch reconstruction

An application with DISCERN and multiple batches based on the pancreas dataset can be found in this Tutorial. This will explain how to use DISCERN with for larger datasets using multiple batches and how to access the effect of different reference batches. The interactive version can be found in Tutorial-MultiBatch.ipynb.

CHAPTER

THREE

CONTRIBUTING

Further contributions to DISCERN are very welcome. To provide a consistent coding style and ensure a running program we implemented tests and code style guidelines.

3.1 Testing

For critical parts of the model several tests has been implemented. They can be run with:

```
poetry run pytest --cov=discern --cov-report=term
```

(Requires the development version of discern).

Some tests are slow and don't run by default, but you can run them using:

```
poetry run pytest --runslow --cov=discern --cov-report=term
```

3.2 Coding style

To enforce code style guidelines pylint and mypy are use. Example commands are shown below:

```
poetry run pylint discern ray_hyperpara.py
poetry run mypy discern ray_hyperpara.py
```

For automatic code formatting yapf was used:

```
yapf -i <filename.py>
```

These tools are included in the dev-dependencies.

CHAPTER

FOUR

API

4.1 DISCERN

DISCERN for expression reconstruction.

`class discern.DISCERN(**kwargs)`

Basic DISCERN model holding a lot of configuration.

Parameters `**kwargs` – DISCERNConfig init args.

`wae_model`

Keras model.

Type Union[None, tf.keras.Model]

`start_step`

Epoch to start training from

Type int

`build_model(n_genes: int, n_labels: int, scale: float)`

Initialize the auto-encoder model and defining the loss and optimizer.

`compile(optimizer: tensorflow.python.keras.optimizer_v2.optimizer_v2.OptimizerV2, scale: float = 15000.0)`

Compile the model and sets losses and metrics.

Parameters

- **`optimizer`** (*`tf.keras.optimizers.Optimizer`*) – Optimizer to use.
- **`scale`** (*`float`*) – Numeric scaling factor for the losses. Defaults to 15000.

`property decoder: tensorflow.python.keras.engine.training.Model`

Return the decoder.

Returns The decoder model.

Return type tf.keras.Model

Raises

- **`ValueError`** – If the decoder is not present.
- **`AttributeError`** – If the model is not build.

`property encoder: tensorflow.python.keras.engine.training.Model`

Return the encoder.

Returns The encoder model.

Return type tf.keras.Model

Raises

- `ValueError` – If the encoder is not present.
- `AttributeError` – If the model is not build.

`classmethod from_json`(*jsondata: Dict[str, Any]*) → DISCERNConfigType

Create an DISCERNConfig instance form hyperparameter json dictionary.

Parameters `jsondata` (*Dict[str, Any]*) – Hyperparameters for this model.

Returns An initialized DISCERNConfig instance

Return type "DISCERNConfig"

Raises `KeyError` – If required key not found in hyperparameter json.

`generate_cells_from_latent`(*latent_codes: Union[tensorflow.python.framework.ops.Tensor, numpy.ndarray]*, *output_batch_labels: Union[tensorflow.python.framework.ops.Tensor, numpy.ndarray]*, *batch_size: int*) → Tuple[numpy.ndarray, numpy.ndarray]

Generate counts from latent codes and batch labels.

Parameters

- `latent_codes` (*Union[tf.Tensor, np.ndarray]*) – Latent codes produced by encoder.
- `output_batch_labels` (*Union[tf.Tensor, np.ndarray]*) – (One Hot) Encoded batch labels for the output. Can also be continous for fuzzy batch association.
- `batch_size` (*int*) – Size of one batch.

Returns the generated count data and dropout probabilities.

Return type Tuple[np.ndarray, np.ndarray]

`generate_latent_codes`(*counts: Union[tensorflow.python.framework.ops.Tensor, numpy.ndarray]*, *batch_labels: Union[tensorflow.python.framework.ops.Tensor, numpy.ndarray]*, *batch_size: int*) → Tuple[numpy.ndarray, numpy.ndarray]

Generate latent codes from count and batch labels.

Parameters

- `counts` (*Union[tf.Tensor, np.ndarray]*) – Count data.
- `batch_labels` (*Union[tf.Tensor, np.ndarray]*) – (One Hot) Encoded batch labels. Can also be continous for fuzzy batch association.
- `batch_size` (*int*) – Size of one batch.

Returns latent codes and sigma values.

Return type Tuple[np.ndarray, np.ndarray]

`get_optimizer`() → tensorflow.python.keras.optimizer_v2.optimizer_v2.OptimizerV2

Create an Optimizer instance.

Returns The created optimizer.

Return type tf.keras.optimizers.Optimizer `project_to_metadata`(*input_data: discern.io.DISCERNData*, *metadata: List[Tuple[str, str]]*, *save_path: pathlib.Path*, *store_sigmas: bool = False*)

Project to average batch with filtering for certain metadata.

Parameters

- **input_data** (`io.DISCERNData`) – Input cells.
- **metadata** (*`List[Tuple[str, str]]`*) – Column-value-Pair used for filerting the cells. Column should match to name in input_data.obs and value to a key in this column.
- **save_path** (*`pathlib.Path`*) – Path for saving the created AnnData objects.
- **store_sigmas** (*`bool, optional`*) – Save sigmas in obsm. Defaults to False.

`reconstruct`(*input_data: discern.io.DISCERNData*, *column: Optional[str]*, *column_value: Optional[str]*, *store_sigmas: bool = False*) → anndata._core.anndata.AnnData

Reconstruct expression data.

Parameters

- **input_data** (`io.DISCERNData`) – DISCERN preprocessed input data
- **column** (*`Optional[str]`*) – Column value used for reconstruction. If *None* just auto-encode the data.
- **column_value** (*`Optional[str], optional`*) – Value in *column* used for reconstruction. If None, project to the average from *column*.
- **store_sigmas** (*`bool, optional`*) – Store latent space sigma values in final output. Defaults to False.

Returns

Reconstructed expression data with input data in raw and DISCERN latent space in obsm.

Return type anndata.AnnData

`restore_model`(*directory: pathlib.Path*)

Restores model from hdf5 checkpoint and compiles it.

Parameters **directory** (*`pathlib.Path`*) – checkpoint directory.

`training`(*inputdata: discern.io.DISCERNData*, *callbacks: Optional[List[tensorflow.python.keras.callbacks.Callback]] = None*, *savepath: Optional[pathlib.Path] = None*, *max_steps: int = 25*) → Dict[str, float]

Train the network *max_steps* times.

Parameters

- **inputdata** (`io.DISCERNData`) – Training data.
- **max_steps** (*`int`*) – Maximum number of epochs to train. Defaults to 25.
- **callbacks** – (List[tf.keras.callbacks.Callback], optional): List of keras callbacks to use. Defaults to None.
- **savepath** (*`pathlib.Path, optional`*) – Filename to save model. Defaults to None.

Returns Metrics from fit method.

Return type Dict[str,float]

`class discern.WAERecipe`(*params: Dict[str, Any], inputs: Optional[Dict[str, anndata._core.anndata.AnnData]] = None, input_files: Optional[Union[Dict[pathlib.Path, str], List[pathlib.Path]]] = None, n_jobs: int = - 1*)

For storing and processing data.

Can apply filtering, clustering. merging and splitting.

Parameters

- **params** (*Dict[str,Any]*) – Default parameters for preprocessing.
- **inputs** (*Dict[str,anndata.AnnData]*) – Input AnnData with batchname as dict-key. Defaults to None.
- **input_files** (*List[pathlib.Path]*) – Paths to raw input data.
- **None.** (*Defaults to*) –
- **n_jobs** (*int*) – Number of jobs/processes to use. Defaults to -1.

sc_raw

Read and concatenated input data.

Type *io.DISCERNData*

config

Parameters calculated during preprocessing.

Type Dict[str, Any]

params

Default parameters for preprocessing.

Type Dict[str,Any]

celltypes()

Aggregate celltype information.

property config

Configuration from preprocessing.

dump(*job_dir: pathlib.Path*)

Dump recipe results to directory.

Parameters **job_dir** (*pathlib.Path*) – The directory to save the results at.

dump_tf_records(*path: pathlib.Path*)

Dump the TFRecords to disk.

Parameters **path** (*pathlib.Path*) – Folder to save the TFrecords in.

filtering(*min_genes: int, min_cells: int*)

Apply filtering in-place.

Parameters

- **min_genes** (*int*) – Minimum number of genes to be present for cell to be considered.
- **min_cells** (*int*) – Minimum number of cells to be present for gene to be considered.

classmethod from_path(*job_dir: pathlib.Path*) → *discern.preprocessing.WAERecipe*

Create WAERecipe from DISCERN directory.

Returns The initalized object.

Return type *WAERecipe*

`kernel_mmd`(*neighbors_mmd: int = 50*, *no_cells_mmd: int = 2000*)

Apply kernel mmd metrics based on nearest neighbors in-place.

Parameters

- **neighbors_mmd** (*int*) – Number of neighbors Defaults to 50.
- **no_cells_mmd** (*int*) – Number of cells used for calculation of mmd. Defaults to 2000.
- **projector** (*Optional[np.ndarray]*) – PCA-Projector to compute distancs in precomputed PCA space. Defaults to None.

`mean_var_scaling`()

Apply Mean-Variance scaling if 'fixed_scaling' is present in params.

`projection_pca`(*pcs: int = 25*)

Apply PCA projection.

Parameters **pcs** (*int*) – Number of principle components. Defaults to 32.

`scaling`(*scale: int*)

Apply scaling in-place.

Parameters **scale** (*int*) – Value use to scale with LSN.

`split`(*split_seed: int*, *valid_cells_ratio: Union[int, float]*, *mmd_cells_ratio: Union[int, float] = 1.0*)

Split cells to train and validation set.

Parameters

- **split_seed** (*int*) – Seed used with numpy.
- **valid_cells_ratio** (*Union[int,float]*) – Number or ratio of cells in the validation set.
- **mmd_cells_ratio** (*Optional[Union[int, float]]*) – Number of validation
- **optimization.** (*cells to use for mmd calculation during hyperparameter*) – Defaults to 1. which is valid_cells_no.

4.2 Reconstruction functions

Basic module containing all functions for running and execution of Model.

class `discern.estimators.DISCERNRunner`(*debug: bool = False*, *gpus: Optional[List[int]] = None*)

Run DISCERN training or project.

Basic DISCERN architecture.

class `discern.estimators.batch_integration.DISCERN`(***kwargs*)

Basic DISCERN model holding a lot of configuration.

Parameters ****kwargs** – DISCERNConfig init args.

`wae_model`

Keras model.

Type Union[None, tf.keras.Model]

`start_step`

Epoch to start training from

Type int

`build_model`(*n_genes: int*, *n_labels: int*, *scale: float*)

Initialize the auto-encoder model and defining the loss and optimizer.

`compile`(*optimizer: tensorflow.python.keras.optimizer_v2.optimizer_v2.OptimizerV2*, *scale: float = 15000.0*)

Compile the model and sets losses and metrics.

Parameters

- `optimizer` (*tf.keras.optimizers.Optimizer*) – Optimizer to use.
- `scale` (*float*) – Numeric scaling factor for the losses. Defaults to 15000.

`property decoder: tensorflow.python.keras.engine.training.Model`

Return the decoder.

Returns The decoder model.

Return type tf.keras.Model

Raises

- `ValueError` – If the decoder is not present.
- `AttributeError` – If the model is not build.

`property encoder: tensorflow.python.keras.engine.training.Model`

Return the encoder.

Returns The encoder model.

Return type tf.keras.Model

Raises

- `ValueError` – If the encoder is not present.
- `AttributeError` – If the model is not build.

`classmethod from_json`(*jsondata: Dict[str, Any]*) → DISCERNConfigType

Create an DISCERNConfig instance form hyperparameter json dictionary.

Parameters `jsondata` (*Dict[str, Any]*) – Hyperparameters for this model.

Returns An initialized DISCERNConfig instance

Return type "DISCERNConfig"

Raises `KeyError` – If required key not found in hyperparameter json.

`generate_cells_from_latent`(*latent_codes: Union[tensorflow.python.framework.ops.Tensor, numpy.ndarray]*, *output_batch_labels: Union[tensorflow.python.framework.ops.Tensor, numpy.ndarray]*, *batch_size: int*) → Tuple[numpy.ndarray, numpy.ndarray]

Generate counts from latent codes and batch labels.

Parameters

- `latent_codes` (*Union[tf.Tensor, np.ndarray]*) – Latent codes produced by encoder.

- **output_batch_labels** (*Union[tf.Tensor, np.ndarray]*) – (One Hot) Encoded batch labels for the output. Can also be continous for fuzzy batch association.
- **batch_size** (*int*) – Size of one batch.

Returns the generated count data and dropout probabilities.

Return type Tuple[np.ndarray, np.ndarray]

`generate_latent_codes`(*counts: Union[tensorflow.python.framework.ops.Tensor, numpy.ndarray]*, *batch_labels: Union[tensorflow.python.framework.ops.Tensor, numpy.ndarray]*, *batch_size: int*) → Tuple[numpy.ndarray, numpy.ndarray]

Generate latent codes from count and batch labels.

Parameters

- **counts** (*Union[tf.Tensor, np.ndarray]*) – Count data.
- **batch_labels** (*Union[tf.Tensor, np.ndarray]*) – (One Hot) Encoded batch labels. Can also be continous for fuzzy batch association.
- **batch_size** (*int*) – Size of one batch.

Returns latent codes and sigma values.

Return type Tuple[np.ndarray, np.ndarray]

`get_optimizer`() → tensorflow.python.keras.optimizer_v2.optimizer_v2.OptimizerV2

Create an Optimizer instance.

Returns The created optimizer.

Return type tf.keras.optimizers.Optimizer

`project_to_metadata`(*input_data:* discern.io.DISCERNData, *metadata: List[Tuple[str, str]]*, *save_path: pathlib.Path*, *store_sigmas: bool = False*)

Project to average batch with filtering for certain metadata.

Parameters

- **input_data** (io.DISCERNData) – Input cells.
- **metadata** (*List[Tuple[str, str]]*) – Column-value-Pair used for filerting the cells. Column should match to name in input_data.obs and value to a key in this column.
- **save_path** (*pathlib.Path*) – Path for saving the created AnnData objects.
- **store_sigmas** (*bool, optional*) – Save sigmas in obsm. Defaults to False.

`reconstruct`(*input_data:* discern.io.DISCERNData, *column: Optional[str]*, *column_value: Optional[str]*, *store_sigmas: bool = False*) → anndata._core.anndata.AnnData

Reconstruct expression data.

Parameters

- **input_data** (io.DISCERNData) – DISCERN preprocessed input data
- **column** (*Optional[str]*) – Column value used for reconstruction. If *None* just auto-encode the data.
- **column_value** (*Optional[str], optional*) – Value in *column* used for reconstruction. If None, project to the average from *column*.
- **store_sigmas** (*bool, optional*) – Store latent space sigma values in final output. Defaults to False.

Returns

Reconstructed expression data with input data in raw and DISCERN latent space in obsm.

Return type anndata.AnnData

`restore_model`(*directory: pathlib.Path*)

Restores model from hdf5 checkpoint and compiles it.

Parameters **`directory`** (*`pathlib.Path`*) – checkpoint directory.

`training`(*inputdata:* discern.io.DISCERNData, *callbacks: Optional[List[tensorflow.python.keras.callbacks.Callback]] = None*, *savepath: Optional[pathlib.Path] = None*, *max_steps: int = 25*) → Dict[str, float]

Train the network *max_steps* times.

Parameters

- **`inputdata`** (`io.DISCERNData`) – Training data.
- **`max_steps`** (*`int`*) – Maximum number of epochs to train. Defaults to 25.
- **`callbacks`** – (List[tf.keras.callbacks.Callback], optional): List of keras callbacks to use. Defaults to None.
- **`savepath`** (*`pathlib.Path, optional`*) – Filename to save model. Defaults to None.

Returns Metrics from fit method.

Return type Dict[str,float]

Module for custom callbacks, especially visualization(UMAP).

`class discern.estimators.callbacks.DelayedEarlyStopping`(*delay: int = 0*, *monitor: str = 'val_loss'*, *min_delta: float = 0.0*, *patience: int = 0*, *verbose: int = 0*, *mode: str = 'auto'*, *baseline: Optional[float] = None*, *restore_best_weights: bool = False*)

Stop when a monitored quantity has stopped improving after some delay time. :param delay: Number of epochs to wait until applying early stopping. :type delay: int :param Defaults to 0: :param which means standard early stopping.: :param monitor: Quantity to be monitored. :type monitor: str :param min_delta: Minimum change in the monitored quantity :type min_delta: float :param to qualify as an improvement: :param i.e. an absolute: :param change of less than min_delta: :param will count as no: :param improvement. Defaults to *val_loss*.: :param patience: Number of epochs with no improvement :type patience: int :param after which training will be stopped. Defaults to 0.: :param verbose: verbosity mode. Defaults to 0. :type verbose: int :param mode: One of *{"auto", "min", "max"}*. In *min* mode, :type mode: str :param training will stop when the quantity: :param monitored has stopped decreasing; in *max*: :param mode it will stop when the quantity: :param monitored has stopped increasing; in *auto*: :param mode: :param the direction is automatically inferred: :param from the name of the monitored quantity. Defaults to *auto*.: :param baseline: Baseline value for the monitored quantity. :type baseline: float, optional :param Training will stop if the model doesn't show improvement over the: :param baseline. Defaults to None.: :param restore_best_weights: Whether to restore model weights from :type restore_best_weights: bool :param the epoch with the best value of the monitored quantity.: :param If False: :param the model weights obtained at the last step of: :param training are used. Defaults to False.:

`on_batch_begin`(*batch*, *logs=None*)

A backwards compatibility alias for *on_train_batch_begin*.

`on_batch_end`(*batch*, *logs=None*)

A backwards compatibility alias for *on_train_batch_end*.

`on_epoch_begin`(*epoch*, *logs=None*)

Called at the start of an epoch.

Subclasses should override for any actions to run. This function should only be called during TRAIN mode.

Parameters

- **`epoch`** – integer, index of epoch.
- **`logs`** – dict. Currently no data is passed to this argument for this method but that may change in the future.

`on_epoch_end`(*epoch: int*, *logs: Optional[Dict[str, Any]] = None*)

Call on epoch end to check for early stopping.

`on_predict_batch_begin`(*batch*, *logs=None*)

Called at the beginning of a batch in *predict* methods.

Subclasses should override for any actions to run.

Parameters

- **`batch`** – integer, index of batch within the current epoch.
- **`logs`** – dict. Has keys *batch* and *size* representing the current batch number and the size of the batch.

`on_predict_batch_end`(*batch*, *logs=None*)

Called at the end of a batch in *predict* methods.

Subclasses should override for any actions to run.

Parameters

- **`batch`** – integer, index of batch within the current epoch.
- **`logs`** – dict. Metric results for this batch.

`on_predict_begin`(*logs=None*)

Called at the beginning of prediction.

Subclasses should override for any actions to run.

Parameters **`logs`** – dict. Currently no data is passed to this argument for this method but that may change in the future.

`on_predict_end`(*logs=None*)

Called at the end of prediction.

Subclasses should override for any actions to run.

Parameters **`logs`** – dict. Currently no data is passed to this argument for this method but that may change in the future.

`on_test_batch_begin`(*batch*, *logs=None*)

Called at the beginning of a batch in *evaluate* methods.

Also called at the beginning of a validation batch in the *fit* methods, if validation data is provided.

Subclasses should override for any actions to run.

Parameters

- **`batch`** – integer, index of batch within the current epoch.

- `logs` – dict. Has keys *batch* and *size* representing the current batch number and the size of the batch.

`on_test_batch_end`(*batch*, *logs=None*)

Called at the end of a batch in *evaluate* methods.

Also called at the end of a validation batch in the *fit* methods, if validation data is provided.

Subclasses should override for any actions to run.

Parameters

- `batch` – integer, index of batch within the current epoch.
- `logs` – dict. Metric results for this batch.

`on_test_begin`(*logs=None*)

Called at the beginning of evaluation or validation.

Subclasses should override for any actions to run.

Parameters `logs` – dict. Currently no data is passed to this argument for this method but that may change in the future.

`on_test_end`(*logs=None*)

Called at the end of evaluation or validation.

Subclasses should override for any actions to run.

Parameters `logs` – dict. Currently no data is passed to this argument for this method but that may change in the future.

`on_train_batch_begin`(*batch*, *logs=None*)

Called at the beginning of a training batch in *fit* methods.

Subclasses should override for any actions to run.

Parameters

- `batch` – integer, index of batch within the current epoch.
- `logs` – dict. Has keys *batch* and *size* representing the current batch number and the size of the batch.

`on_train_batch_end`(*batch*, *logs=None*)

Called at the end of a training batch in *fit* methods.

Subclasses should override for any actions to run.

Parameters

- `batch` – integer, index of batch within the current epoch.
- `logs` – dict. Metric results for this batch.

`on_train_begin`(*logs=None*)

Called at the beginning of training.

Subclasses should override for any actions to run.

Parameters `logs` – dict. Currently no data is passed to this argument for this method but that may change in the future.

`on_train_end`(*logs=None*)

Called at the end of training.

Subclasses should override for any actions to run.

Parameters `logs` – dict. Currently no data is passed to this argument for this method but that may change in the future.

class `discern.estimators.callbacks.VisualisationCallback`(*outdir: Union[str, pathlib.Path]*, *data: anndata._core.anndata.AnnData*, *batch_size: int*, *freq: int = 10*)

Redo prediction on datasets and visualize via UMAP.

Parameters

- **outdir** (*pathlib.Path*) – Output directory for the figures.
- **data** (*anndata.AnnData*) – Input cells.
- **batch_size** (*int*) – Numer of cells to visualize.
- **freq** (*int*) – Frequency for computing visualisations in epochs. Defaults 10.

`on_batch_begin`(*batch*, *logs=None*)

A backwards compatibility alias for *on_train_batch_begin*.

`on_batch_end`(*batch*, *logs=None*)

A backwards compatibility alias for *on_train_batch_end*.

`on_epoch_begin`(*epoch*, *logs=None*)

Called at the start of an epoch.

Subclasses should override for any actions to run. This function should only be called during TRAIN mode.

Parameters

- **epoch** – integer, index of epoch.
- **logs** – dict. Currently no data is passed to this argument for this method but that may change in the future.

`on_epoch_end`(*epoch: int*, *logs: Optional[Dict[str, float]] = None*)

Run on epoch end. Executes only at specified frequency.

Parameters

- **epoch** (*int*) – Epochnumber.
- **logs** (*Optional[Dict[str, float]]*) – losses and metrics passed by tensorflow fit . Defaults to None.

`on_predict_batch_begin`(*batch*, *logs=None*)

Called at the beginning of a batch in *predict* methods.

Subclasses should override for any actions to run.

Parameters

- **batch** – integer, index of batch within the current epoch.
- **logs** – dict. Has keys *batch* and *size* representing the current batch number and the size of the batch.

`on_predict_batch_end`(*batch*, *logs=None*)

Called at the end of a batch in *predict* methods.

Subclasses should override for any actions to run.

Parameters

- `batch` – integer, index of batch within the current epoch.
- `logs` – dict. Metric results for this batch.

`on_predict_begin`(*logs=None*)

Called at the beginning of prediction.

Subclasses should override for any actions to run.

Parameters `logs` – dict. Currently no data is passed to this argument for this method but that may change in the future.

`on_predict_end`(*logs=None*)

Called at the end of prediction.

Subclasses should override for any actions to run.

Parameters `logs` – dict. Currently no data is passed to this argument for this method but that may change in the future.

`on_test_batch_begin`(*batch*, *logs=None*)

Called at the beginning of a batch in *evaluate* methods.

Also called at the beginning of a validation batch in the *fit* methods, if validation data is provided.

Subclasses should override for any actions to run.

Parameters

- `batch` – integer, index of batch within the current epoch.
- `logs` – dict. Has keys *batch* and *size* representing the current batch number and the size of the batch.

`on_test_batch_end`(*batch*, *logs=None*)

Called at the end of a batch in *evaluate* methods.

Also called at the end of a validation batch in the *fit* methods, if validation data is provided.

Subclasses should override for any actions to run.

Parameters

- `batch` – integer, index of batch within the current epoch.
- `logs` – dict. Metric results for this batch.

`on_test_begin`(*logs=None*)

Called at the beginning of evaluation or validation.

Subclasses should override for any actions to run.

Parameters `logs` – dict. Currently no data is passed to this argument for this method but that may change in the future.

on_test_end(*logs=None*)

Called at the end of evaluation or validation.

Subclasses should override for any actions to run.

Parameters **logs** – dict. Currently no data is passed to this argument for this method but that may change in the future.

on_train_batch_begin(*batch*, *logs=None*)

Called at the beginning of a training batch in *fit* methods.

Subclasses should override for any actions to run.

Parameters

- **batch** – integer, index of batch within the current epoch.
- **logs** – dict. Has keys *batch* and *size* representing the current batch number and the size of the batch.

on_train_batch_end(*batch*, *logs=None*)

Called at the end of a training batch in *fit* methods.

Subclasses should override for any actions to run.

Parameters

- **batch** – integer, index of batch within the current epoch.
- **logs** – dict. Metric results for this batch.

on_train_begin(*logs: Optional[Dict[str, float]] = None*)

Run on training start.

Parameters **logs** (`Optional[Dict[str, float]]`) – logs, not used only for compatibility reasons.

on_train_end(*logs: Optional[Dict[str, float]] = None*)

Run on training end.

Parameters **logs** (`Optional[Dict[str, float]]`) – losses and metrics passed by tensorflow fit . Defaults to None.

discern.estimators.callbacks.**create_callbacks**(*early_stopping_limits: Dict[str, Any]*, *exp_folder: pathlib.Path*, *inputdata: Optional[*discern.io.DISCERNData*] = None*, *umap_cells_no: Optional[int] = None*, *profile_batch: int = 2*, *freq_of_viz: int = 30*) → List[tensorflow.python.keras.callbacks.Callback]

Generate list of callbacks used by tensorflow model.fit.

Parameters

- **early_stopping_limits** (`Dict[str,Any]`) – Patience, min_delta, and delay for early stopping.
- **exp_folder** (`str`) – Folder where everything is saved.
- **inputdata** (`io.DISCERNData, optional`) – Input data to use. Defaults to None
- **umap_cells_no** (`int`) – Number of cells for UMAP.
- **profile_batch** (`int`) – Number of the batch to do extensive profiling. Defaults to 2. (see tf.keras.callbacks.Tensorboard)

- **freq_of_viz** (*int*) – Frequency of visualization callback in epochs. Defaults to 30.

Returns callbacks used by tensorflow model.fit.

Return type List[callbacks.Callback]

Custom Keras Layers.

class discern.estimators.customlayers.**GaussianReparametrization**(*trainable=True*, *name=None*, *dtype=None*, *dynamic=False*, ***kwargs*)

Reparametrization layer using gaussians.

build(*input_shape: Tuple[tensorflow.python.framework.ops.Tensor, tensorflow.python.framework.ops.Tensor]*)

Build the layer, usually automatically called at first call.

Parameters **input_shape** (*Tuple[tf.Tensor, tf.Tensor]*) – Shape of the inputs. Both should have as last dimension the size of the latent space.

static **call**(*inputs: Tuple[tensorflow.python.framework.ops.Tensor, tensorflow.python.framework.ops.Tensor]*, ***kwargs: Dict[str, Any]*) → tensorflow.python.framework.ops.Tensor

Call the layer.

Parameters

- **inputs** (*Tuple[tf.Tensor, tf.Tensor]*) – latent codes and sigmas from encoder
- ****kwargs** (*Dict[str,Any]*) – Additional attributes, should contain 'training'".

Returns Rescaled latent codes.

Return type tf.Tensor

class discern.estimators.customlayers.**MMDPP**(*scale: float*, ***kwargs*)

mmdpp penalty calculation in keras layer.

Parameters **scale** (*float*) – Value used to scale the output.

scale

Value used to scale the output.

Type float

build(*input_shape: Tuple[tensorflow.python.framework.ops.Tensor, tensorflow.python.framework.ops.Tensor]*)

Build the layer, usually automatically called at first call.

Parameters **input_shape** (*Tuple[tf.Tensor, tf.Tensor]*) – Shape of the inputs. Both shapes should have the size of the latent space as last dimension.

call(*inputs: Tuple[tensorflow.python.framework.ops.Tensor, tensorflow.python.framework.ops.Tensor]*, ***kwargs: Dict[str, Any]*) → tensorflow.python.framework.ops.Tensor

Call the layer.

Parameters **inputs** (*Tuple[tf.Tensor, tf.Tensor]*) – The latent codes and sigma values from encoder.

Returns mmdpp penalty loss.

Return type tf.Tensor

get_config() → Dict[str, Any]

Return configuration of the layer. Used for serialization.

Returns Configuration of the layer

Return type Dict[str,Any]

class discern.estimators.customlayers.**SigmaRegularization**(*trainable=True*, *name=None*, *dtype=None*, *dynamic=False*, ***kwargs*)

Regularization term to push sigmas near to one.

build(*input_shape: tensorflow.python.framework.ops.Tensor*)

Build the layer, usually automatically called at first call.

Parameters **input_shape** (*tf.Tensor*) – Shape of the input.

call(*inputs: tensorflow.python.framework.ops.Tensor*, ***kwargs: Dict[str, Any]*) → tensorflow.python.framework.ops.Tensor

Call the layer.

Parameters **inputs** (*tf.Tensor*) – Inputs to layer consisting of sigma values.

Returns Regularization loss

Return type tf.Tensor

discern.estimators.customlayers.**condlayernorm**(*input_cells: tensorflow.python.framework.ops.Tensor*, *labels: tensorflow.python.framework.ops.Tensor*, *size: int*, *regularization: Optional[Dict[str, Any]] = None*) → tensorflow.python.framework.ops.Tensor

Create a conditioning layer.

Parameters

- **input_cells** (*tf.Tensor*) – Input to the laxer
- **labels** (*tf.Tensor*) – Label for each sample.
- **size** (*int*) – Size of the output/input

Returns

The output of the conditioning layer, with the same size as the input and spezified in size.

Return type tf.Tensor

discern.estimators.customlayers.**getmembers**() → Dict[str, tensorflow.python.keras.engine.base_layer.Layer]

Return a dictionary of all custom layers defined in this module.

Returns Name and class of custom layers.

Return type Dict[str, tf.keras.layers.Layer]

discern.estimators.customlayers.**mmdpp_penalty**(*sample_qz: tensorflow.python.framework.ops.Tensor*, *sample_pz: tensorflow.python.framework.ops.Tensor*, *encoder_sigma: tensorflow.python.framework.ops.Tensor*, *total_number_cells: float*, *latent_dim: int*) → tensorflow.python.framework.ops.Tensor

Calculate the mmdpp penalty.

Based on https://github.com/tolstikhin/wae/blob/master/improved_wae.py

Parameters

- **sample_qz** (*tf.Tensor*) – Sample from the aggregated posterior.
- **sample_pz** (*tf.Tensor*) – Sample from the prior.
- **encoder_sigma** (*tf.Tensor*) – Sigma values from the random encoder.
- **total_number_cells** (*int*) – Total number of samples for scaling.
- **latent_dim** (*int*) – Dimension of the latent space.

Returns mmdpp penalty loss.

Return type tf.Tensor

Module containing all losses.

`class discern.estimators.losses.`**`DummyLoss`**`(`*reduction: int = 'auto', name: str = 'Dummy'*`)`

Dummy loss simpy passing the input y_pred as loss output.

Parameters

- **reduction** (*int*) – Reduction type to use. Defaults to tf.keras.losses.Reduction.AUTO.
- **name** (*str*) – Name of the loss. Defaults to 'Dummy'.

`static` **`call`**`(`*y_true, y_pred*`)`

Call the loss and returns the predicted value.

`classmethod` **`from_config`**`(`*config*`)`

Instantiates a *Loss* from its config (output of *get_config()*).

Parameters **config** – Output of *get_config()*.

Returns A *Loss* instance.

`class discern.estimators.losses.`**`HuberLoss`**`(`*delta=1.0, reduction='auto', name='huber_loss'*`)`

Huber loss.

`call``(`*y_true, y_pred*`)`

Calculate Huber loss.

`classmethod` **`from_config`**`(`*config*`)`

Instantiates a *Loss* from its config (output of *get_config()*).

Parameters **config** – Output of *get_config()*.

Returns A *Loss* instance.

`class discern.estimators.losses.`**`Lnorm`**`(`*p: int, name: str = 'LNorm', reduction: str = 'auto', axis: int = 0, epsilon: float = 1e-20, use_root: bool = False*`)`

Calculate the Lnorm of input and output.

Parameters

- **p** (*int*) – Which Lnorm to calculate, for example p=1 means L1-Norm.
- **name** (*str*) – Description of parameter *name*. Defaults to 'LNorm'.
- **reduction** (*int*) – Reduction type to use. Defaults to tf.keras.losses.Reduction.AUTO.
- **axis** (*int*) – Axis on which the norm is calculated. Defaults to 0.
- **epsilon** (*float*) – Small value to add if (square)root is used.. Defaults to 1e-20.
- **use_root** (*bool*) – Use (square)root. Defaults to False.

`pnorm`
Which Lnorm to calculate, for example p=1 means L1-Norm.

Type int

`epsilon`
Small value to add if (square)root is used.

Type float

`axis`
Axis on which the norm is calculated.

Type int

`use_root`
Use (square)root.

Type bool

`call`(*y_true*, *y_pred*)
Call and returns the loss.

`classmethod from_config`(*config*)
Instantiates a *Loss* from its config (output of *get_config()*).

Parameters `config` – Output of *get_config()*.

Returns A *Loss* instance.

`get_config`()
Serialize the loss.

`class discern.estimators.losses.MaskedCrossEntropy`(*zeros: numpy.ndarray*, *zeros_eps: float = 1e-06*, *lower_label_smoothing: float = 0.0*, ***kwargs*)
Categorical crossentropy Loss with creates mask in true data.

Parameters

- `zeros` (*np.ndarray*) – Value(s) which represent values to be zeros.
- `zeros_eps` (*float*) – Value to check for approximate matching to *zeros*.

`call`(*y_true: tensorflow.python.framework.ops.Tensor*, *y_pred: tensorflow.python.framework.ops.Tensor*) → tensorflow.python.framework.ops.Tensor
Call of the loss.

`classmethod from_config`(*config*)
Instantiates a *Loss* from its config (output of *get_config()*).

Parameters `config` – Output of *get_config()*.

Returns A *Loss* instance.

`get_config`()
Return the configuration of the loss.

`discern.estimators.losses.getmembers`() → Dict[str, Union[tensorflow.python.keras.losses.Loss, tensorflow.python.keras.metrics.Metric]]
Return a dictionary of all custom losses and metrics defined in this module.

Returns Name and class of custom losses and metrics.

Return type Dict[str, Union[tf.keras.losses.Loss,tf.keras.metrics.Metric]]

`discern.estimators.losses.`**`reconstruction_loss`**(*loss_type: Dict[str, Any]*) → tensorflow.python.framework.ops.Tensor

Generate different loss classes based on dictionary.

Parameters **`loss_type`** (*`Dict[str, Any]`*) – Dictionary with name as classname of the loss and all parameter to be set.

Returns Calculated loss (object)

Return type tf.Tensor

Raises **`KeyError`** – When the loss name is not supported.

Basic module for running an experiment.

`class discern.estimators.run_exp.`**`CheckMetaData`**(*dataframe: pandas.core.frame.DataFrame*)

Check MetaData column value pair in dataframe lazy.

`check`(*metadata_tuple: List[str]*) → Tuple[str, str]

Check if column value pair is present.

Parameters **`metadata_tuple`** (*`List[str]`*) – Column value pair

Returns Input column value pair.

Return type Tuple[str, str]

`class discern.estimators.run_exp.`**`DISCERNRunner`**(*debug: bool = False*, *gpus: Optional[List[int]] = None*)

Run DISCERN training or project.

`discern.estimators.run_exp.`**`run_exp_multiprocess`**(*exp_folder: pathlib.Path*, *available_gpus: List[int]*, *func: Callable[..., None]*, *kwargs: Optional[Dict[str, Any]] = None*) → int

Run an experiment with forced GPU setting (suitable for python mp).

Parameters

- **`exp_folder`** (*`pathlib.Path`*) – Path to the experiement.
- **`available_gpus`** (*`List[int]`*) – List of available GPUs.
- **`func`** (*`Callable[..., None]`*) – Train or eval function.
- **`kwargs`** (*`Optional[Dict[str, Any]]`*) – Additional arguments passed to the called functions. Defaults to None.

Returns Status code, 0 is success, 1 is failure.

Return type int

`discern.estimators.run_exp.`**`run_projection`**(*exp_folder: pathlib.Path*, *metadata: List[str]*, *infile: Optional[Union[str, pathlib.Path]]*, *all_batches: bool*, *store_sigmas: bool*)

Run projection to metadata on trained model.

Parameters

- **`exp_folder`** (*`pathlib.Path`*) – Folder/ Experiment name to the trained model.
- **`metadata`** (*`List[str]`*) – Metadata to use for integration. Should be like List[*column name:value,...*]

- **infile** (*Optional[Union[str, pathlib.Path]]*) – Alternative input file.
- **all_batches** – (bool): Project to all batches.
- **store_sigmas** – (bool): Store sigmas after projection.

discern.estimators.run_exp.**run_train**(*exp_folder: pathlib.Path*, *input_path: Optional[pathlib.Path] = None*)

Run an experiment.

Parameters

- **exp_folder** (*pathlib.Path*) – Experiments folders.
- **input_path** (*Optional[pathlib.Path]*) – Input path for the TFRecords, if None the experiments folder is used. Defaults to None.

discern.estimators.run_exp.**setup_exp**(*exp_folder: pathlib.Path*) → Tuple[*discern.estimators.batch_integration.DISCERN*, Dict[str, Any]]

Setup experiment, by assigning the GPU and parsing the model.

Parameters **exp_folder** (*pathlib.Path*) – Experiment folder.

Returns The model, the output path for training and all parameters.

Return type Tuple[*batch_integration.DISCERN*, pathlib.Path, Dict[str, Any]]

A number of classes and functions used across all types of models.

discern.estimators.utilities_wae.**create_decoder**(*latent_dim: int*, *output_cells_dim: int*, *dec_layers: List[int]*, *dec_norm_type: List[str]*, *activation_fn: Callable[tensorflow.python.framework.ops.Tensor, tensorflow.python.framework.ops.Tensor]*, *output_fn: Optional[str]*, *n_labels: int*, *regularization: float*, *output_lsn: Optional[float] = None*, *conditional_regularization: Optional[Dict[str, Any]] = None*) → tensorflow.python.keras.engine.training.Model

Create a decoder.

Parameters

- **latent_dim** (*int*) – Dimension of the latent space.
- **output_cells_dim** (*int*) – Dimension of the output.
- **dec_layers** (*List[int]*) – Dimensions for the decoder layers.
- **dec_norm_type** (*List[str]*) – Normalization type, eg. BatchNormalization.
- **activation_fn** (*Callable[[tf.Tensor], tf.Tensor]*) – Activation function in the model.
- **output_fn** (*str*) – Function to produce gene counts.
- **n_labels** (*int*) – Number of labels for the batch labels.
- **regularization** (*float*) – Dropout rate.
- **output_lsn** (*Optional[float]*) – Scaling parameter, used for softmax and LSN.

Returns The decoder.

Return type tf.keras.Model

`discern.estimators.utilities_wae.`**`create_encoder`**(*latent_dim: int*, *enc_layers: List[int]*, *enc_norm_type: List[str]*, *activation_fn: Callable[tensorflow.python.framework.ops.Tensor, tensorflow.python.framework.ops.Tensor]*, *input_dim: int*, *n_labels: int*, *regularization: float*, *conditional_regularization: Optional[Dict[str, Any]] = None*) → tensorflow.python.keras.engine.training.Model

Create an Encoder.

Parameters

- **latent_dim** (*int*) – Dimension of the latent space.
- **enc_layers** (*List[int]*) – Dimension of the encoding layers.
- **enc_norm_type** (*List[str]*) – Normalization type, eg. BatchNormalization.
- **activation_fn** (*Callable[[tf.Tensor], tf.Tensor]*) – Activation function in the model.
- **input_dim** (*int*) – Dimension of the input.
- **n_labels** (*int*) – Number of labels for the batch labels.
- **regularization** (*float*) – Rate of dropout.

Returns The encoder.

Return type tf.keras.Model

Raises **NotImplementedError** – If enc_norm_type is not understood.

`discern.estimators.utilities_wae.`**`create_model`**(*encoder: tensorflow.python.keras.engine.training.Model*, *decoder: tensorflow.python.keras.engine.training.Model*, *total_number_cells: float*, *name: str = 'WAE'*) → tensorflow.python.keras.engine.training.Model

Generate a model from encoder and decoder, adding gaussian noise (reparametrization).

Parameters

- **encoder** (*tf.keras.Model*) – The encoder.
- **decoder** (*tf.keras.Model*) – The decoder.
- **total_number_cells** (*int*) – Total number of cells used for scaling MMDPP.
- **name** (*str*) – Name of the model. Defaults to "WAE".

Returns The created model including SigmaRegularization and MMDPP loss.

Return type tf.keras.Model

`discern.estimators.utilities_wae.`**`load_model_from_directory`**(*directory: pathlib.Path*) → Tuple[Union[None, tensorflow.python.keras.engine.training.Model], int]

Load model from latest checkpoint using its hdf5 file.

Parameters **directory** (*pathlib.Path*) – Name of the directory with hdf5 files.

Returns

Full model and last step. None and zero if no models could be loaded.

Return type Tuple[Union[None, tf.keras.Model], int]

4.3 I/O functions

discern i/o operations.

`class discern.io.DISCERNData`(*adata: anndata._core.anndata.AnnData*, *batch_size: int*, *cachefile: Optional[Union[str, pathlib.Path]] = ''*)

DISCERNData for storing and reading inputs.

`property batch_size: int`

Get batch size.

`property config: Dict[str, Any]`

Get DISCERN data dependent configuration.

`classmethod from_folder`(*folder: pathlib.Path*, *batch_size: int*, *cachefile: Optional[Union[str, pathlib.Path]] = ''*) → *discern.io.DISCERNData*

Read data from DISCERN folder.

Returns The data including AnnData and TFRecords.

Return type *DISCERNData*

`classmethod read_h5ad`(*filename: pathlib.Path*, *batch_size: int*, *cachefile: Optional[Union[str, pathlib.Path]] = ''*) → *discern.io.DISCERNData*

Create DISCERNData from anndata H5AD file.

Returns The single cell data.

Return type *DISCERNData*

`property tfdata: Tuple[tensorflow.python.data.ops.dataset_ops.DatasetV2, tensorflow.python.data.ops.dataset_ops.DatasetV2]`

The accociated tf.data.Datasets.

Returns Training and validation data.

Return type Tuple[tf.data.Dataset, tf.data.Dataset]

`property zeros`

Get Zero representation in current data.

`class discern.io.TFRecordsWriter`(*out_dir: pathlib.Path*)

Context manager to be used for writing tf.data.Dataset to TFRecord file.

Parameters `out_dir` (*pathlib.Path*) – Path to the directory where to write the TFRecords.

`out_dir`

Path to the directory where to write the TFRecords.

Type str

`write_dataset`(*dataset: tensorflow.python.data.ops.dataset_ops.DatasetV2*, *split: str*)

Write tf.data.Dataset to TFRecord specified by split.

Parameters

- `dataset` (*tf.data.Dataset*) – Dataset to be written.

- `split` (*str*) – Subfile to use: *train* or *valid*.

Raises `ValueError` – If split is not supported.

`discern.io.`**`estimate_csr_nbytes`**`(`*mat: numpy.ndarray*`)` → int

Estimates the size of a sparse matrix generated from numpy array.

Parameters `mat` (*np.ndarray*) – Input array.

Returns Estimated size of the sparse matrix.

Return type int

`discern.io.`**`generate_h5ad`**`(`*counts: Union[numpy.ndarray, Tuple[numpy.ndarray, numpy.ndarray]], var: pandas.core.frame.DataFrame, obs: pandas.core.frame.DataFrame, save_path: Optional[pathlib.Path] = None, threshold: float = 0.1, **kwargs*`)` → anndata._core.anndata.AnnData

Generate AnnData format and can save it to file.

Parameters

- `counts` (*Union[np.ndarray, Tuple[np.ndarray, np.ndarray]]*) – Count data (X in AnnData).
- `var` (*pd.DataFrame*) – Variables dataframe.
- `obs` (*pd.DataFrame*) – Observations dataframe.
- `save_path` (*Optional[pathlib.Path]*) – Save path for the AnnData in h5py file. Defaults to None.
- `threshold` (*float*) – Set values lower than *threshold* to zero. Defaults to 0.1.
- `kwargs` – Keyword arguments passed to anndata.AnnData.

Returns The AnnData file.

Return type anndata.AnnData

`discern.io.`**`make_dataset_from_anndata`**`(`*adata: anndata._core.anndata.AnnData, for_tfrecord: bool = False*`)` → Tuple[tensorflow.python.data.ops.dataset_ops.DatasetV2, tensorflow.python.data.ops.dataset_ops.DatasetV2]

Generate TensorFlow Dataset from AnnData object.

Parameters

- `adata` (*anndata.AnnData*) – Input cells
- `for_tfrecord` (*bool*) – make output for writing TFrecords. Defaults to False.

Returns The training and validation datasets

Return type Tuple[tf.data.Dataset, tf.data.Dataset]

`discern.io.`**`np_one_hot`**`(`*labels: pandas.core.arrays.categorical.Categorical*`)` → numpy.ndarray

One hot encode a numpy array.

Parameters `labels` (*pd.Categorical*) – integer values used as indices

Returns One hot encoded labels

Return type np.ndarray

`discern.io.parse_tfrecords`(*tfr_files: Union[pathlib.Path, List[pathlib.Path]]*, *genes_no: int*, *n_labels: int*) → tensorflow.python.data.ops.dataset_ops.DatasetV2

Generate TensorFlow dataset from TensorFlow records file(s).

Parameters

- **tfr_files** (*Union[pathlib.Path, List[pathlib.Path]]*) – TFRecord file(s).
- **genes_no** (*int*) – Number of genes in the TFRecords.
- **n_labels** (*int*) – Number of batch labels
- **batch_size** (*int*) – Size of one batch

Returns

Dataset containing 'input_data', 'batch_input_enc' and 'batch_input_dec'

Return type tf.data.Dataset

4.4 Preprocessing functions

Contains the GeneMatrix class, used to represent the scRNA-seq data.

class discern.preprocessing.**WAERecipe**(*params: Dict[str, Any]*, *inputs: Optional[Dict[str, anndata._core.anndata.AnnData]] = None*, *input_files: Optional[Union[Dict[pathlib.Path, str], List[pathlib.Path]]] = None*, *n_jobs: int = - 1*)

For storing and processing data.

Can apply filtering, clustering. merging and splitting.

Parameters

- **params** (*Dict[str,Any]*) – Default parameters for preprocessing.
- **inputs** (*Dict[str,anndata.AnnData]*) – Input AnnData with batchname as dict-key. Defaults to None.
- **input_files** (*List[pathlib.Path]*) – Paths to raw input data.
- **None.** (*Defaults to*) –
- **n_jobs** (*int*) – Number of jobs/processes to use. Defaults to -1.

sc_raw

Read and concatenated input data.

Type *io.DISCERNData*

config

Parameters calculated during preprocessing.

Type Dict[str, Any]

params

Default parameters for preprocessing.

Type Dict[str,Any]

celltypes()

Aggregate celltype information.

property config

Configuration from preprocessing.

dump(*job_dir: pathlib.Path*)

Dump recipe results to directory.

Parameters **job_dir** (*pathlib.Path*) – The directory to save the results at.

dump_tf_records(*path: pathlib.Path*)

Dump the TFRecords to disk.

Parameters **path** (*pathlib.Path*) – Folder to save the TFrecords in.

filtering(*min_genes: int*, *min_cells: int*)

Apply filtering in-place.

Parameters

- **min_genes** (*int*) – Minimum number of genes to be present for cell to be considered.
- **min_cells** (*int*) – Minimum number of cells to be present for gene to be considered.

classmethod from_path(*job_dir: pathlib.Path*) → *discern.preprocessing.WAERecipe*

Create WAERecipe from DISCERN directory.

Returns The initalized object.

Return type *WAERecipe*

kernel_mmd(*neighbors_mmd: int = 50*, *no_cells_mmd: int = 2000*)

Apply kernel mmd metrics based on nearest neighbors in-place.

Parameters

- **neighbors_mmd** (*int*) – Number of neighbors Defaults to 50.
- **no_cells_mmd** (*int*) – Number of cells used for calculation of mmd. Defaults to 2000.
- **projector** (*Optional[np.ndarray]*) – PCA-Projector to compute distancs in precomputed PCA space. Defaults to None.

mean_var_scaling()

Apply Mean-Variance scaling if 'fixed_scaling' is present in params.

projection_pca(*pcs: int = 25*)

Apply PCA projection.

Parameters **pcs** (*int*) – Number of principle components. Defaults to 32.

scaling(*scale: int*)

Apply scaling in-place.

Parameters **scale** (*int*) – Value use to scale with LSN.

split(*split_seed: int*, *valid_cells_ratio: Union[int, float]*, *mmd_cells_ratio: Union[int, float] = 1.0*)

Split cells to train and validation set.

Parameters

- **split_seed** (*int*) – Seed used with numpy.
- **valid_cells_ratio** (*Union[int,float]*) – Number or ratio of cells in the validation set.
- **mmd_cells_ratio** (*Optional[Union[int, float]]*) – Number of validation

- **optimization.** (*cells to use for mmd calculation during hyperparameter*) – Defaults to 1. which is valid_cells_no.

`discern.preprocessing.`**`merge_data_sets`**(*raw_inputs: Dict[str, anndata._core.anndata.AnnData]*, *batch_keys: Dict[str, str]*) → Tuple[anndata._core.anndata.AnnData, Dict[int, str]]

Merge a dictionary of AnnData files to a single AnnData object.

Parameters **`raw_inputs`** (*`Dict[str, anndata.AnnData]`*) – Names and AnnData objects.

Returns Merged AnnData and mapping from codes to names.

Return type Tuple[anndata.AnnData, Dict[int, str]]

`discern.preprocessing.`**`read_process_serialize`**(*job_path: pathlib.Path*, *with_tfrecords: bool = True*)

Read data, preprocesses it and write output as anndata.AnnData and TFRecords.

Parameters

- **`job_path`** (*`pathlib.Path`*) – Path of the experiments folder.
- **`with_tfrecords`** (*`bool`*) – write tfrecord files. Defaults to True.

`discern.preprocessing.`**`read_raw_input`**(*file_path: pathlib.Path*) → anndata._core.anndata.AnnData

Read input and converts it to anndata.AnnData object.

Currently h5, h5ad, loom, txt and a directory with matrix.mtx. genes.tsv and optional barcodes.tsv is supported.

Parameters **`file_path`** (*`pathlib.Path`*) – (File-) Path to the input data.

Returns The read AnnData object.

Return type anndata.AnnData

Raises **`ValueError`** – Datatype of input could not be interfered.

4.5 Online/Incremental learning

Module for supporting online learning.

class `discern.online_learning.`**`OnlineDISCERNRunner`**(*debug: bool*, *gpus: List[int]*)

DISCERNRunner supporting online learning.

class `discern.online_learning.`**`OnlineWAERecipe`**(*reference_adata:* discern.io.DISCERNData, **args*, ***kwargs*)

WAERecipe for the online setting.

This class allows the same preprocessing as done for reference data, which allows further processing in using DISCERN online learning.

`celltypes`()

Aggregate celltype information.

property config

Configuration from preprocessing.

`dump`(*job_dir: pathlib.Path*)

Dump recipe results to directory.

Parameters **`job_dir`** (*`pathlib.Path`*) – The directory to save the results at.

`dump_tf_records`(*path: pathlib.Path*)

Dump the TFRecords to disk.

Parameters `path` (*pathlib.Path*) – Folder to save the TFrecords in.

`filtering`(*min_genes: int*, **unused_args*, ***unused_kwargs*)

Apply filtering in-place.

Parameters `min_genes` (*int*) – Minimum number of genes to be present for cell to be considered.

`fix_batch_labels`()

Fix batch labels codes by including old categories.

`classmethod from_path`(*job_dir: pathlib.Path*) → *discern.preprocessing.WAERecipe*

Create WAERecipe from DISCERN directory.

Returns The initalized object.

Return type *WAERecipe*

`kernel_mmd`(*neighbors_mmd: int = 50*, *no_cells_mmd: int = 2000*)

Apply kernel mmd metrics based on nearest neighbors in-place.

Parameters

- `neighbors_mmd` (*int*) – Number of neighbors Defaults to 50.
- `no_cells_mmd` (*int*) – Number of cells used for calculation of mmd. Defaults to 2000.
- `projector` (*Optional[np.ndarray]*) – PCA-Projector to compute distancs in precomputed PCA space. Defaults to None.

`mean_var_scaling`()

Apply Mean-Variance scaling if 'fixed_scaling' is present.

`projection_pca`(*pcs: int = 25*)

Apply PCA projection from reference.

Parameters `pcs` (*int*) – Number of principle components. Defaults to 32.

`scaling`(*scale: int*)

Apply scaling in-place.

Parameters `scale` (*int*) – Value use to scale with LSN.

`split`(*split_seed: int*, *valid_cells_ratio: Union[int, float]*, *mmd_cells_ratio: Union[int, float] = 1.0*)

Split cells to train and validation set.

Parameters

- `split_seed` (*int*) – Seed used with numpy.
- `valid_cells_ratio` (*Union[int,float]*) – Number or ratio of cells in the validation set.
- `mmd_cells_ratio` (*Optional[Union[int, float]]*) – Number of validation
- `optimization.` (*cells to use for mmd calculation during hyperparameter*) – Defaults to 1. which is valid_cells_no.

discern.online_learning.**online_training**(*exp_folder: pathlib.Path*, *filename: pathlib.Path*, *freeze: bool*)

Continue running an experiment.

Parameters

- **exp_folder** (*pathlib.Path*) – Experiments folders.
- **filename** (*pathlib.Path*) – Input path for new data set.
- **freeze** (*bool*) – Freeze non conditional layers.

discern.online_learning.**save_data**(*file: pathlib.Path*, *data:* discern.io.DISCERNData, *old_data:* discern.io.DISCERNData)

Save data by concatenating to reference.

discern.online_learning.**update_model**(*old_model: tensorflow.python.keras.engine.training.Model*, *new_model: tensorflow.python.keras.engine.training.Model*, *freeze_unchanged: bool = False*) → tensorflow.python.keras.engine.training.Model

Update the weights from an old model to a new model.

New model can have a bigger weight size for the layers in the first dimension.

Parameters

- **old_model** (*tf.keras.Model*) – Old, possibly trained model.
- **new_model** (*tf.keras.Model*) – New model, for which the weights should be set (inplace).
- **freeze_unchanged** (*bool, optional*) – Freeze layers in the new model, which weights didn't changed in size compared to the old model. Defaults to False.

Returns The updated new model.

Return type tf.keras.Model

4.6 Other functions

Module containing diverse TensorFlow related functions.

discern.functions.**get_function_by_name**(*func_str: str*) → Callable[..., Any]

Get a function by its name.

Parameters **func_str** (*str*) – Name of function including module, like 'tensorflow.nn.softplus'.

Returns the function.

Return type Callable[Any]

Raises **KeyError** – Function does not exists.

discern.functions.**getmembers**(*name: str*) → Dict[str, Any]

Return a dictionary of all classes defined in this module.

Parameters **name** (*str*) – Name of the module. Usually __name__.

Returns Name and class of module.

Return type Dict[str,Any]

`discern.functions.`**`parse_mean_var`**(*features: pandas.core.frame.DataFrame*, *scalings: Dict[str, Union[str, float]]*) → Tuple[numpy.ndarray, numpy.ndarray]

Get mean and variance from anndata.var and scaling dict.

Parameters

- **`features`** (*`pd.DataFrame`*) – anndata.var.
- **`scalings`** – (Dict[str, Union[str, float]]): Scalings dict.

Returns Mean and variance

Return type Tuple[np.ndarray, np.ndarray]

`discern.functions.`**`prepare_train_valid`**(*input_tfr: pathlib.Path*) → Tuple[pathlib.Path, pathlib.Path]

Get all filennames for train and validation files.

Parameters **`input_tfr`** (*`pathlib.Path`*) – Name of input directory.

Returns

Add train and validation files to seperate lists.

Return type Tuple[List[pathlib.Path], List[pathlib.Path]]

`discern.functions.`**`rescale_by_params`**(*adata: anndata._core.anndata.AnnData*, *scalings: Dict[str, Union[str, float, int]]*) → anndata._core.anndata.AnnData

Rescale counts by fixed mean and variance (inplace).

Reverting function scale_by_params.

Parameters

- **`adata`** (*`anndata.AnnData`*) – Data to be rescaled.
- **`scalings`** (*`Dict[str, Union[str, float, int]]`*) – Mean and scale values used for rescaling. Can be numeric or *genes*. *genes* means using precomputed values in anndata object like *adata.var['mean_scaling']* and *adata.var['var_scaling']*, respectively.

Raises **`ValueError`** – mean and variance not numeric or *genes*.

Returns The rescaled AnnData object

Return type anndata.AnnData

`discern.functions.`**`sample_counts`**(*counts: numpy.ndarray*, *probabilities: numpy.ndarray*, *var: Optional[pandas.core.frame.DataFrame] = None*, *uns: Optional[Dict[str, Any]] = None*) → numpy.ndarray

Sample counts using probabilities.

Parameters

- **`counts`** (*`np.ndarray`*) – Count data.
- **`probabilities`** (*`np.ndarray`*) – Probability of being non-zero.

Returns Sampled count data.

Return type np.ndarray

`discern.functions.`**`scale`**(*adata: anndata._core.anndata.AnnData*, *mean: Optional[Union[numpy.ndarray, float]] = None*, *var: Optional[Union[numpy.ndarray, float]] = None*) → anndata._core.anndata.AnnData

Scale counts by fixed mean and variance.

Parameters

- `adata` (*anndata.AnnData*) – Data to be scaled.
- `mean` (*Optional[np.ndarray]*) – Mean for scaling (will be zero-centered). Defaults to None
- `var` (*Optional[np.ndarray]*) – Variance for scaling (will be rescaled to 1). Defaults to None

Returns The AnnData file.

Return type anndata.AnnData

`discern.functions.`**`scale_by_params`**(*adata: anndata._core.anndata.AnnData*, *scalings: Dict[str, Union[str, float]]*) → anndata._core.anndata.AnnData

Scale counts by fixed mean and variance (inplace).

Parameters

- `adata` (*anndata.AnnData*) – Data to be scaled.
- `scalings` (*Dict[str, Union[str, float]]*) – Mean and scale values used for scaling. Can be numeric or *genes*. *genes* means using precomputed values in anndata object like *adata.var['mean_scaling']* and *adata.var['var_scaling']*, respectively.

Raises `ValueError` – mean and variance not numeric or *genes*.

Returns The scaled AnnData object

Return type anndata.AnnData

`discern.functions.`**`set_gpu_and_threads`**(*n_threads: int*, *gpus: Optional[List[int]]*)

Limits CPU and GPU usage.

Parameters

- `n_threads` (*int*) – Number of threads to use (get splittet to inter- and intra-op threads). Can be disabled by feeding 0.
- `gpus` (*List[int]*) – List of GPUs to use. Use all GPUs by passing None and no GPUs by passing an empty list.

Module for fast MMD calculations.

`discern.mmd.`**`mmd_loss`**(*random_cells: numpy.ndarray*, *valid_cells: numpy.ndarray*, *sigma: float*) → float

Compute mmd loss between random cells and valid cells.

Parameters

- `random_cells` (*np.ndarray*) – Random generated cells.
- `valid_cells` (*np.ndarray*) – Valid (decoded) cells.
- `sigma` (*float*) – Precalculated Sigma value.

Returns MMD loss between random and valid cells.

Return type float

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX