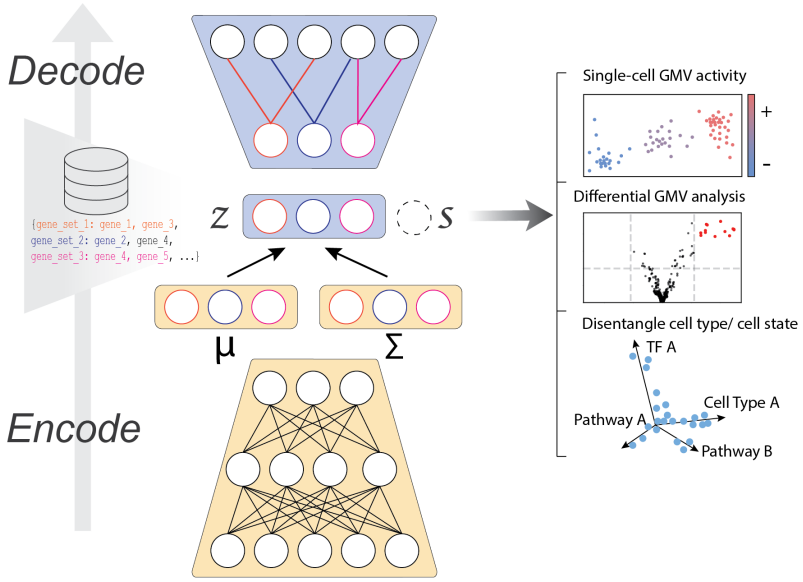
VEGA

Lucas Seninge, Ioannis Anastopoulos

Apr 19, 2022

CONTENTS

VEGA is a deep generative model for scRNA-Seq data whose decoder structure is informed by gene modules such as pathways, gene regulatory networks, or cell type marker sets. It allows embedding of single-cell data into an interpretable latent space, inference of gene module activity at the single-cell level, and differential activity testing for those gene modules between groups of cells. VEGA is implemented in Pytorch and works around the scanpy and scvi-tools ecosystems.



CHAPTER

ONE

GETTING STARTED

VEGA simply requires

- An Anndata single-cell dataset
- GMT file(s) with the gene module membership, such as provided by databases like MSigDB

Note: We recommend to preprocess the single-cell dataset before passing it to VEGA.

CHAPTER

TWO

MAIN FEATURES

VEGA provides the following features

- Embed single-cell data into an interpretable latent space
- Inference of gene module activities at the single-cell level
- Cell type / cell state disentanglement
- Alternative to enrichment methods for finding differentially activated pathways

Differential pathway activity Inspired by the differential gene expression procedure from `scvi-tools`, VEGA provides a Bayesian testing procedure to find significantly activated gene modules in your dataset. More information in the vega basic usage tutorial.

CHAPTER

THREE

NAVIGATE THE DOCS

3.1 Installation

Before installing VEGA, we recommend to make a dedicated virtual environment, for example with conda:

```
conda create -n vega python=3.7.0
```

3.1.1 With PyPI

To install with `pip`, simply run:

```
pip install scvega
```

3.2 API

Import vega with:

```
import vega
```

3.2.1 VEGA

class vega.**VEGA**(*adata*, *gmt_paths=None*, *add_nodes=1*, *min_genes=0*, *max_genes=5000*, *positive_decoder=True*, *encode_covariates=False*, *regularizer='mask'*, *reg_kwargs=None*, ***kwargs*)

Constructor for class VEGA (VAE Enhanced by Gene Annotations).

Parameters

- **adata** (`AnnData`) – scanpy single-cell object. Please run setup_anndata() before passing to VEGA.
- **gmt_paths** (`Union[str, list, None]`) – one or more paths to .gmt files for GMVs initialization.
- **add_nodes** (`int`) – additional fully-connected nodes in the mask.
- **min_genes** (`int`) – minimum gene size for GMVs.
- **max_genes** (`int`) – maximum gene size for GMVs.

- **positive_decoder** (bool) – whether to constrain decoder to positive weights
- **encode_covariates** (bool) – whether to encode covariates along gene expression
- **regularizer** (str) – which regularization strategy to use (l1, gelnet, mask). Default: mask.
- **reg_kwargs** (Optional[dict]) – parameters for regularizer.
- ****kwargs** –

  use_cuda using CPU (False) or CUDA (True).

  beta weight for KL-divergence.

  dropout dropout rate in model

  z_dropout dropout rate for the latent space (for correlation).

save(*path*, *save_adata=False*, *save_history=False*, *overwrite=False*, *save_regularizer_kwargs=True*)
Save model parameters to input directory. Saving Anndata object and training history is optional.

Parameters

- **path** (str) – path to save directory
- **save_adata** (bool) – whether to save the Anndata object in the save directory
- **save_history** (bool) – whether to save the training history in the save directory
- **save_regularizer_kwargs** (bool) – whether to save regularizer hyperparameters (lambda, penalty matrix…) in the save directory

classmethod load(*path*, *adata=None*, *device=device(type='cpu')*, *reg_kwargs=None*)
Load model from directory. If adata=None, try to reload Anndata object from saved directory.

Parameters

- **path** (str) – path to save directory
- **adata** (Optional[AnnData]) – scanpy single cell object
- **device** (device) – CPU or CUDA

encode(*X*, *batch_index*, *cat_covs=None*)
Encode data in latent space (Inference step).

Parameters

- **X** – input data
- **batch_index** – batch information for samples
- **cat_covs** – categorical covariates

Returns

- *z* – data in latent space
- *mu* – mean of variational posterior
- *logvar* – log-variance of variational posterior

decode(*z*, *batch_index*, *cat_covs=None*)
Decode data from latent space.

Parameters

- **z** – data embedded in latent space
- **batch_index** – batch information for samples

- **cat_covs** – categorical covariates.

Returns decoded data

Return type X_rec

sample_latent(*mu*, *logvar*)
Sample latent space with reparametrization trick. First convert to std, sample normal(0,1) and get Z.

Parameters

- **mu** – mean of variational posterior
- **logvar** – log-variance of variational posterior

Returns sampled latent space

Return type eps

to_latent(*adata=None*, *indices=None*, *return_mean=False*)
Project data into latent space. Inspired by SCVI.

Parameters

- **adata** (Optional[AnnData]) – scanpy single-cell dataset
- **indices** (Optional[list]) – indices of the subset of cells to be encoded
- **return_mean** (bool) – whether to use the mean of the multivariate gaussian or samples

generative(*adata=None*, *indices=None*, *use_mean=True*)
Generate new samples from input data (encode-decode).

Parameters

- **adata** (Optional[AnnData]) – scanpy single-cell dataset
- **indices** (Optional[list]) – indices of the subset of cells to be encoded
- **use_mean** (bool) – whether to use the mean of the multivariate gaussian or samples

differential_activity(*groupby*, *adata=None*, *group1=None*, *group2=None*, *mode='change'*, *delta=2.0*, *fdr_target=0.05*, ***kwargs*)
Bayesian differential activity procedures for GMVs. Similar to scVI [Lopez2018] Bayesian DGE but for latent variables. Differential results are saved in the adata object and returned as a DataFrame.

Parameters

- **groupby** (str) – anndata object field to group cells (eg. *"cell type"*)
- **adata** (Optional[AnnData]) – scanpy single-cell object. If None, use Anndata attribute of VEGA.
- **group1** (Union[str, list, None]) – reference group(s).
- **group2** (Union[str, list, None]) – outgroup(s).
- **mode** (str) – differential activity mode. If *"vanilla"*, uses [Lopez2018], if *"change"* uses [Boyeau2019].
- **delta** (float) – differential activity threshold for *"change"* mode.
- **fdr_target** (float) – minimum FDR to consider gene as DE.
- ****kwargs** – optional arguments of the bayesian_differential method.

Returns

Return type Differential activity results

bayesian_differential(*adata*, *cell_idx1*, *cell_idx2*, *n_samples=5000*, *use_permutations=True*, *n_permutations=5000*, *mode='change'*, *delta=2.0*, *alpha=0.66*, *random_seed=False*)

Run Bayesian differential expression in latent space. Returns Bayes factor of all factors.

Parameters

- **adata** (AnnData) – anndata single-cell object.
- **cell_idx1** (list) – indices of group 1.
- **cell_idx2** (list) – indices of group 2.
- **n_samples** (int) – number of samples to draw from the latent space.
- **use_permutations** (bool) – whether to use permutations when computing the double integral.
- **n_permutations** (int) – number of permutations for MC integral.
- **mode** (int) – differential activity test strategy. *"vanilla"* corresponds to [Lopez2018], *"change"* to [Boyeau2019].
- **delta** (float) – for mode *"change"*, the differential threshold to be used.
- **random_seed** (bool) – seed for reproducibility.

Returns dictionary with results (Bayes Factor, Mean Absolute Difference)

Return type res

forward(*tensors*)

Forward pass through full network.

Parameters **tensors** – input data

Returns dictionary of output tensors

Return type out_tensors

vae_loss(*model_input*, *model_output*)

Custom loss for beta-VAE

Parameters

- **model_input** – dict with input values
- **model_output** – dict with output values

Returns

Return type loss value for current batch

train_vega(*learning_rate=0.0001*, *n_epochs=500*, *train_size=1.0*, *batch_size=128*, *shuffle=True*, *use_gpu=False*, ***kwargs*)

Main method to train VEGA.

Parameters

- **learning_rate** (float) – learning rate
- **n_epochs** (int) – number of epochs to train model
- **train_size** (float) – a number between 0 and 1 to indicate the proportion of training data. Test size is set to 1-train_size
- **batch_size** (int) – number of samples per batch

- **shuffle** (bool) – whether to shuffle samples or not
- **use_gpu** (bool) – whether to use GPU
- ****kwargs** – other keyword arguments of the _train_model() method, like the early stopping patience

3.2.2 VegaSCVI

class vega.**VegaSCVI**(*adata*, *gmt_paths=None*, *add_nodes=1*, *min_genes=0*, *max_genes=5000*, *positive_decoder=True*, *n_hidden=600*, *n_layers=2*, *gene_likelihood='zinb'*, *dropout_rate=0.1*, *z_dropout=0*, *use_cuda=True*, ***model_kwargs*)

VEGA: VAE Enhanced by Gene Annotations [Seninge2021].

Parameters

- **adata** – AnnData object that has been registered via `setup_anndata()`.
- **gmt_paths** – A single or list of paths to .GMT files with gene annotations for GMVs initialization.
- **add_nodes** – Number of additional fully-connected decoder nodes (unannotated GMVs).
- **min_genes** – Minimum gene size for GMVs.
- **max_genes** – Maximum gene size for GMVs.
- **positive_decoder** – Whether to constrain decoder to positive weights.
- **n_hidden** – Number of nodes per hidden layer.
- **n_layers** – Number of hidden layers used for encoder NN.
- **gene_likelihood** – Likelihood function for the generative model.
- **dropout_rate** – Dropout rate for neural networks.
- **use_cuda** – Using GPU with CUDA

3.2.3 Regularizers

class vega.regularizers.**GelNet**(*lambda1*, *lambda2*, *P*, *d=None*, *lr=0.001*, *use_gpu=False*)

GelNet regularizer for linear decoder [Sokolov2016]. If `P` is set to Identity matrix, this is Elastic net. `d` needs to be a *{0,1}*-matrix. If `lamda1` is 0, this is a L2 regularization. If `lambda2` is 0, this is a L1 regularization.

Needs to be sequentially used in training loop.

Example

```
>>> loss = MSE(X_hat, X)
# Compute L2 term
>>> loss += GelNet.quadratic_update(self.decoder.weight)
>>> loss.backward()
>>> optimizer.step()
# L1 proximal operator update
>>> GelNet.proximal_update(self.decoder.weight)
```

Parameters

- **lambda1** (float) – L1-regularization coefficient

- **lambda2** (float) – L2-regularization coefficient
- **P** (ndarray) – Penalty matrix (eg. Gene network Laplacian)
- **d** (Optional[ndarray]) – Domain knowledge matrix (eg. mask)
- **lr** (float) – Learning rate

quadratic_update(*weights*)
Computes the L2 term of GelNet

Parameters **weights** – Layer's weight matrix

proximal_update(*weights*)
Proximal operator for the L1 term inducing sparsity.

Parameters **weights** – Layer's weight matrix

class vega.regularizers.**LassoRegularizer**(*lambda1*, *lr*, *d=None*, *use_gpu=False*)
Lasso (L1) regularizer for linear decoder. Similar to [Rybakov2020] lasso regularization.

Parameters

- **lambda1** (float) – L1-regularization coefficient
- **d** (Optional[ndarray]) – Domain knowledge matrix (eg. mask)
- **lr** (float) – Learning rate

quadratic_update(*weights*)
Not applicable (identity)

proximal_update(*weights*)
Proximal operator for the L1 term inducing sparsity.

Parameters **weights** – Layer's weight matrix

3.2.4 Utils functions

vega.utils.**setup_anndata**(*adata*, *batch_key=None*, *categorical_covariate_keys=None*, *copy=False*)
Creates VEGA fields in input Anndata object for mask. Also creates SCVI field which will be used for batch and covariates.

Parameters

- **adata** (AnnData) – Scanpy single-cell object
- **copy** (bool) – Whether to return a copy or change in place
- **batch_key** (Optional[str]) – Observation to be used as batch
- **categorical_covariate_keys** (Union[str, list, None]) – Observation to use as covariate keys

Returns updated object if copy is True

Return type adata

vega.utils.**create_mask**(*adata*, *gmt_paths=None*, *add_nodes=1*, *min_genes=0*, *max_genes=1000*, *copy=False*)
Initialize mask M for GMV from one or multiple .gmt files.

Parameters

- **adata** (AnnData) – Scanpy single-cell object.

- **gmt_paths** (Union[str, list, None]) – One or several paths to .gmt files.
- **add_nodes** (int) – Additional latent nodes for capturing additional variance.
- **min_genes** (int) – Minimum number of genes per GMV.
- **max_genes** (int) – Maximum number of genes per GMV.
- **copy** (bool) – Whether to return a copy of the updated Anndata object.

Returns Scanpy single-cell object.

Return type adata

vega.utils.**preprocess_anndata**(*adata*, *n_top_genes=5000*, *copy=False*)

Simple (default) Scanpy preprocessing function before autoencoders.

Parameters

- **adata** (AnnData) – Scanpy single-cell object
- **n_top_genes** (int) – Number of highly variable genes to retain
- **copy** (bool) – Return a copy or in place

Returns Preprocessed Anndata object

Return type adata

3.2.5 Plotting functions

vega.plotting.**volcano**(*adata*, *group1*, *group2*, *sig_lvl=3.0*, *metric_lvl=3.0*, *annotate_gmv=None*, *s=10*, *fontsize=10*, *textsize=8*, *figsize=None*, *title=False*, *save=False*)

Plot Differential GMV results. Please run the Bayesian differential acitvity method of VEGA before plotting ("model.differential_activity()")

Parameters

- **adata** (AnnData) – scanpy single-cell object
- **group1** (str) – name of reference group
- **group2** (str) – name of out-group
- **sig_lvl** (float) – absolute Bayes Factor cutoff (>=0)
- **metric_lvl** (float) – mean Absolute Difference cutoff (>=0)
- **annotate_gmv** (Union[str, list, None]) – GMV to be displayed. If None, all GMVs passing significance thresholds are displayed
- **s** (int) – dot size
- **fontsize** (int) – text size for axis
- **textsize** (int) – text size for GMV name display
- **title** (str) – title for plot
- **save** (Union[str, bool]) – path to save figure as pdf

vega.plotting.**gmv_embedding**(*adata*, *x*, *y*, *color=None*, *palette=None*, *title=None*, *save=False*, *sct_kwds=None*)

2-D scatter plot in GMV space.

Parameters

- **adata** (AnnData) – scanpy single-cell object. VEGA analysis needs to be run before
- **x** (str) – GMV name for x-coordinates (eg. 'REACTOME_INTERFERON_SIGNALING')
- **y** (str) – GMV name for y-coordinates (eg. 'REACTOME_INTERFERON_SIGNALING')
- **color** (Optional[str]) – categorical field of Anndata.obs to color single-cells
- **title** (Optional[str]) – plot title
- **save** (Union[str, bool]) – path to save plot
- **sct_kwds** (Optional[dict]) – kwargs for matplotlib.pyplot.scatter function

vega.plotting.**gmv_plot**(*adata*, *x*, *y*, *color=None*, *title=None*, *palette=None*)
GMV embedding plot, but using the Scanpy plotting API.

Parameters

- **adata** (AnnData) – scanpy single-cell dataset
- **x** (str) – GMV name for x-coordinates (eg. 'REACTOME_INTERFERON_SIGNALING')
- **y** (str) – GMV name for x-coordinates (eg. 'REACTOME_INTERFERON_SIGNALING')
- **color** (Optional[str]) – .obs field to color by
- **title** (Optional[str]) – title for the plot
- **palette** (Optional[str]) – matplotlib colormap to be used

vega.plotting.**loss**(*model*, *plot_validation=True*)
Plot training loss and validation if plot_validation is True.

Parameters

- **model** (*VEGA*) – VEGA model (trained)
- **plot_validation** (bool) – Whether to plot validation loss as well

vega.plotting.**rank_gene_weights**(*model*, *gmv_list*, *n_genes=10*, *color_in_set=True*, *n_panels_per_row=3*, *fontsize=8*, *star_names=[]*, *save=False*)
Plot gene members of input GMVs according to their magnitude (abs(w)). Inspired by scanpy.pl.rank_gene_groups() API.

Parameters

- **model** (*VEGA*) – VEGA trained model
- **gmv_list** (Union[str, list]) – list of GMV names
- **n_genes** (int) – number of top gene to display
- **color_in_set** (bool) – Whether to color genes annotated as part of GMVs differently.
- **n_panels_per_row** (int) – number of panels max. per row
- **star_names** (list) – Name of genes to be highlighted with stars
- **save** (Union[bool, str]) – path to save figure

vega.plotting.**weight_heatmap**(*model*, *cluster=True*, *cmap='viridis'*, *display_gmvs='all'*, *display_genes='all'*, *title=None*, *figsize=None*, *save=False*, *hm_kwargs=None*)
Heatmap plots of weights.

Parameters

- **model** (*VEGA*) – VEGA trained model

- **cluster** (bool) – if True, use hierarchical clustering (seaborn.clustermap)
- **cmap** (str) – colormap to use
- **display_gmvs** (Union[str, list]) – if all, display all latent variables weights. Else (list) only the subset
- **display_genes** (Union[str, list]) – if all, display all gene weights of GMV. Else (list) only the subset
- **title** (Optional[str]) – figure title
- **figsize** (Union[tuple, list, None]) – figure size
- **save** (Union[bool, str]) – path to save figure
- **hm_kwargs** (Optional[dict]) – kwargs for sns.clustermap or sns.heatmap (depending on if cluster=True)

3.3 Tutorials

For basic usage of VEGA on training the model and analyzing pathway activities (including differential tests)

3.3.1 Analysis of Reactome pathway activities in PBMCs with VEGA

In this tutorial, we will demonstrate a use case for VEGA on single-cell PBMCs from Kang et al. The dataset is composed of 2 conditions (cells stimulated with interferon-beta and control cells). We will analyze the GMVs (Gene Module Variables) activities of the two populations.

[1]:
```
# Load VEGA and packages
import sys
import vega
import pandas as pd
import scanpy as sc
# Ignore warnings in this tutorial
import warnings
warnings.filterwarnings('ignore')
```

Loading and preparing data

We will load the PBMC data. The data were already preprocessed. As shown, the data is comprised of 16893 cells with 6998 highly variable genes.

[2]:
```
adata = vega.data.pbmc()
print(adata)
```

```
AnnData object with n_obs × n_vars = 16893 × 6998
    obs: 'condition', 'n_counts', 'n_genes', 'mt_frac', 'cell_type'
    var: 'gene_symbol', 'n_cells'
    uns: 'cell_type_colors', 'condition_colors', 'neighbors'
    obsm: 'X_pca', 'X_tsne', 'X_umap'
    obsp: 'distances', 'connectivities'
```

VEGA is built around the `scvi-tools` ecosystem. As such, we need to run our own version of `setup_anndata`, which in addition to locating covariates and important additional information also initializes attributes in the Anndata object that will be used by VEGA.

[3]:
```
vega.utils.setup_anndata(adata)
```

```
Running VEGA and SCVI setup...
INFO     No batch_key inputted, assuming all cells are same batch
INFO     No label_key inputted, assuming all cells have same label
INFO     Using data from adata.X
INFO     Computing library size prior per batch
INFO     Successfully registered anndata object containing 16893 cells, 6998 vars, 1␣
↪batches,
         1 labels, and 0 proteins. Also registered 0 extra categorical covariates and 0␣
↪extra
         continuous covariates.
INFO     Please do not further modify adata until model is trained.
```

Creating and training the VEGA model

Similarly to `scvi-tools`, it is very straightforward to create and train VEGA models. In addition to specifying the input Anndata object, we also specify the path to `.gmt` files that will be used to initialize VEGA's latent space, as well as the number of unnannotated variables to model with the `add_nodes` argument. Additionally, we force the weights of VEGA's linear decoder to be positive to aid interpretability when running differential activity tests later.

Note 1: The model is initialized with a masked decoder as described in the original VEGA paper, but other regularization strategies are available (l1, ElasticNet, GelNet...)*

Note 2: Change the `gmt_paths` variable to match the location of your `.gmt` file. REACTOME file can be downloaded here

[4]:
```
model = vega.VEGA(adata,
                  gmt_paths='reactomes.gmt',
                  add_nodes=1,
                  positive_decoder=True)
print(model)
```

```
Using masked decoder
Constraining decoder to positive weights
VEGA model with the following parameters:
n_GMVs: 675, dropout_rate:0.1, z_dropout:0.3, beta:5e-05, positive_decoder:True
Model is trained: False
```

[5]:
```
model.train_vega(n_epochs=50)
```

```
[Epoch 1] | loss: 331.439 |
[Epoch 2] | loss: 236.868 |
[Epoch 3] | loss: 210.888 |
[Epoch 4] | loss: 203.683 |
[Epoch 5] | loss: 200.081 |
[Epoch 6] | loss: 197.597 |
[Epoch 7] | loss: 195.700 |
[Epoch 8] | loss: 194.076 |
[Epoch 9] | loss: 192.903 |
[Epoch 10] | loss: 191.478 |
```

(continues on next page)

(continued from previous page)

```
[Epoch 11] | loss: 190.245 |
[Epoch 12] | loss: 189.227 |
[Epoch 13] | loss: 188.143 |
[Epoch 14] | loss: 187.366 |
[Epoch 15] | loss: 186.178 |
[Epoch 16] | loss: 185.538 |
[Epoch 17] | loss: 184.696 |
[Epoch 18] | loss: 183.901 |
[Epoch 19] | loss: 183.318 |
[Epoch 20] | loss: 182.713 |
[Epoch 21] | loss: 182.158 |
[Epoch 22] | loss: 181.271 |
[Epoch 23] | loss: 180.924 |
[Epoch 24] | loss: 180.391 |
[Epoch 25] | loss: 179.938 |
[Epoch 26] | loss: 179.593 |
[Epoch 27] | loss: 179.010 |
[Epoch 28] | loss: 178.464 |
[Epoch 29] | loss: 178.168 |
[Epoch 30] | loss: 177.891 |
[Epoch 31] | loss: 177.364 |
[Epoch 32] | loss: 177.091 |
[Epoch 33] | loss: 176.875 |
[Epoch 34] | loss: 176.007 |
[Epoch 35] | loss: 175.734 |
[Epoch 36] | loss: 175.610 |
[Epoch 37] | loss: 175.454 |
[Epoch 38] | loss: 174.753 |
[Epoch 39] | loss: 174.666 |
[Epoch 40] | loss: 174.361 |
[Epoch 41] | loss: 174.108 |
[Epoch 42] | loss: 173.987 |
[Epoch 43] | loss: 173.325 |
[Epoch 44] | loss: 173.272 |
[Epoch 45] | loss: 173.043 |
[Epoch 46] | loss: 172.780 |
[Epoch 47] | loss: 172.768 |
[Epoch 48] | loss: 172.313 |
[Epoch 49] | loss: 172.292 |
[Epoch 50] | loss: 171.890 |
```

Saving and reloading

To save and reload a trained VEGA model, simply use the following syntax. To verify that a model is trained, print the model variable.

[6]:
```
#model.save('./vega_pbmc/', save_adata=True, save_history=True)
```

[6]:
```
#model = vega.VEGA.load('.vega_pbmc/')
print(model)
```

```
VEGA model with the following parameters:
n_GMVs: 675, dropout_rate:0.1, z_dropout:0.3, beta:5e-05, positive_decoder:True
Model is trained: True
```

Analyzing model output

One primary feature of VEGA is to project the transcriptomes into an interpretable latent space representing the gene sets used to initalize the model.

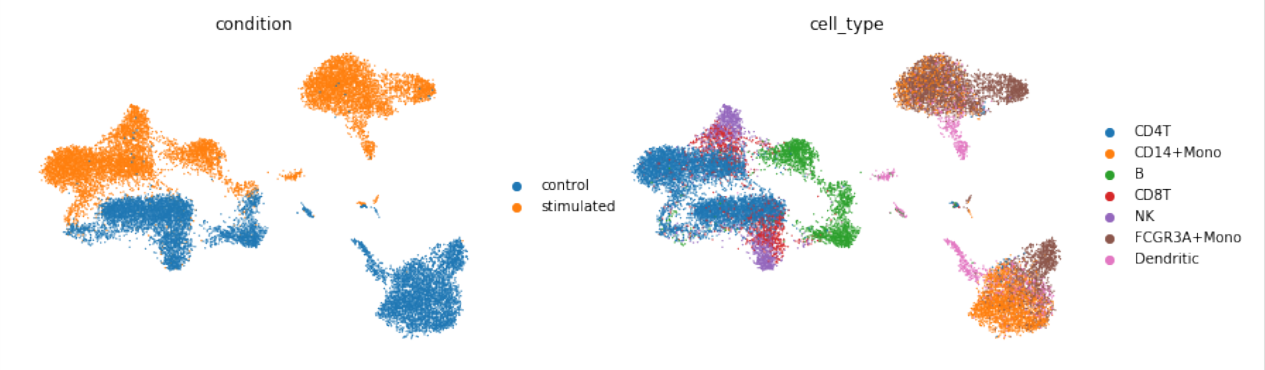
[7]:
```
model.adata.obsm['X_vega'] = model.to_latent()
```

UMAP with VEGA latent space

We can now use the interoperability with Scanpy to get a UMAP representation of the data based on VEGA's latent space.

[8]:
```
sc.pp.neighbors(model.adata, use_rep='X_vega', n_neighbors=15)
sc.tl.umap(model.adata, min_dist=0.5, random_state=42)
```

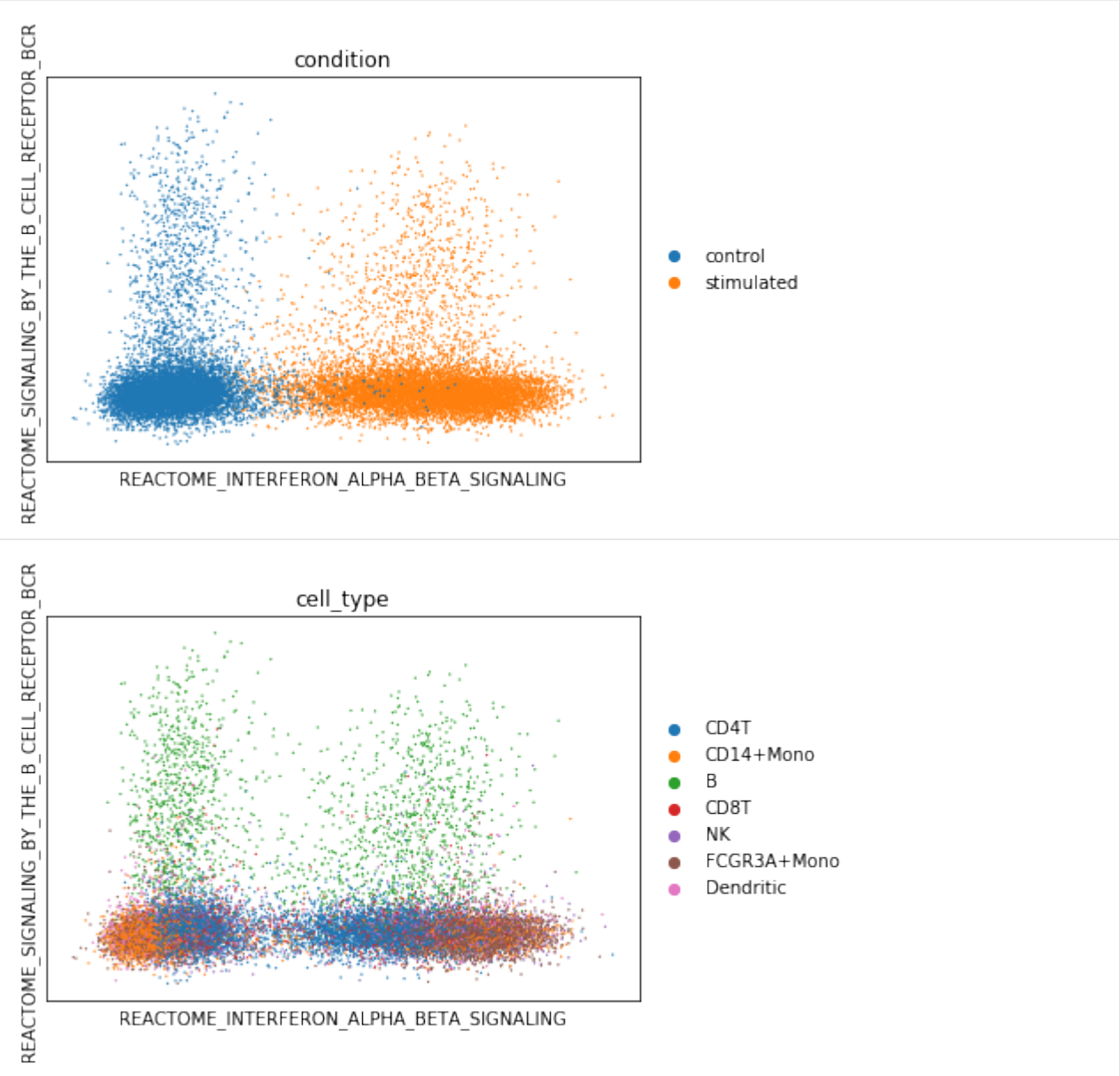
[9]:
```
sc.pl.umap(model.adata, color=["condition", "cell_type"], frameon=False)
```



Analyzing GMV activities

We can now project cells into individual pathway components for biological processes of interest. Here, we are interested to see if we can use the interferon pathway activity to separate the 2 conditions. We expect interferon pathways to be activated in the *'stimulated'* condition. Also, we use the BCR signaling pathway to segregate B-cells from the rest of the dataset.

[11]:
```
vega.plotting.gmv_plot(model.adata,
                       x='REACTOME_INTERFERON_ALPHA_BETA_SIGNALING',
                       y='REACTOME_SIGNALING_BY_THE_B_CELL_RECEPTOR_BCR',
                       color='condition')
vega.plotting.gmv_plot(model.adata,
                       x='REACTOME_INTERFERON_ALPHA_BETA_SIGNALING',
                       y='REACTOME_SIGNALING_BY_THE_B_CELL_RECEPTOR_BCR',
                       color='cell_type')
```



Differential activity testing between groups

Similarly to differential expression analysis in scRNA-Seq, we can use VEGA's latent space for testing the difference in pathway activities between 2 groups of cells. This provides an interesting alternative to enrichment tests such as performed by GSEA.

[12]:
```
da_df = model.differential_activity(groupby='condition', fdr_target=0.1)
da_df.head()
```

```
Using VEGA's adata attribute for differential analysis
No reference group: running 1-vs-rest analysis for .obs[condition]
```

[12]:

	p_da	p_not_da	\
REACTOME_INTERFERON_SIGNALING	0.9852	0.0148	

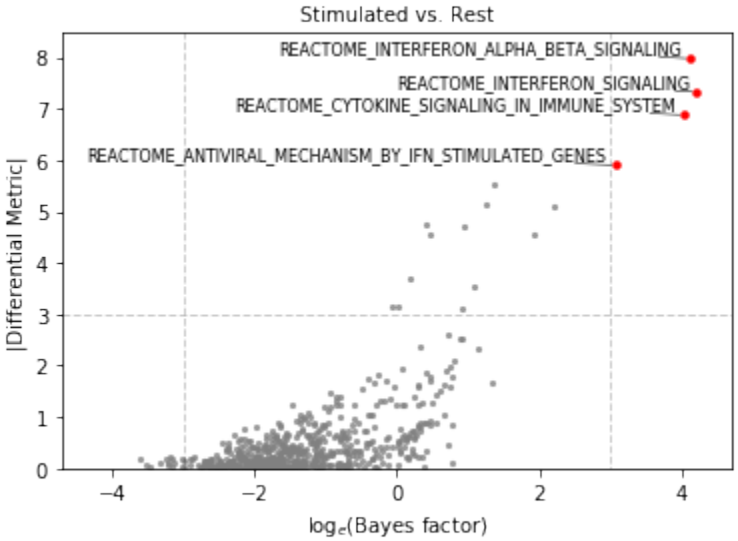
(continues on next page)

(continued from previous page)

```
REACTOME_INTERFERON_ALPHA_BETA_SIGNALING             0.9838    0.0162
REACTOME_CYTOKINE_SIGNALING_IN_IMMUNE_SYSTEM         0.9824    0.0176
REACTOME_ANTIVIRAL_MECHANISM_BY_IFN_STIMULATED_...   0.9554    0.0446
REACTOME_RIG_I_MDA5_MEDIATED_INDUCTION_OF_IFN_A...   0.9018    0.0982

                                                    bayes_factor  \
REACTOME_INTERFERON_SIGNALING                           4.198217
REACTOME_INTERFERON_ALPHA_BETA_SIGNALING                4.106411
REACTOME_CYTOKINE_SIGNALING_IN_IMMUNE_SYSTEM            4.022100
REACTOME_ANTIVIRAL_MECHANISM_BY_IFN_STIMULATED_...      3.064396
REACTOME_RIG_I_MDA5_MEDIATED_INDUCTION_OF_IFN_A...      2.217387

                                                    is_da_alpha_0.66  \
REACTOME_INTERFERON_SIGNALING                                   True
REACTOME_INTERFERON_ALPHA_BETA_SIGNALING                        True
REACTOME_CYTOKINE_SIGNALING_IN_IMMUNE_SYSTEM                    True
REACTOME_ANTIVIRAL_MECHANISM_BY_IFN_STIMULATED_...              True
REACTOME_RIG_I_MDA5_MEDIATED_INDUCTION_OF_IFN_A...              True

                                                    differential_metric  \
REACTOME_INTERFERON_SIGNALING                                  7.322754
REACTOME_INTERFERON_ALPHA_BETA_SIGNALING                       7.976952
REACTOME_CYTOKINE_SIGNALING_IN_IMMUNE_SYSTEM                   6.879438
REACTOME_ANTIVIRAL_MECHANISM_BY_IFN_STIMULATED_...             5.902566
REACTOME_RIG_I_MDA5_MEDIATED_INDUCTION_OF_IFN_A...             5.115499

                                                    delta  is_da_fdr_0.1  \
REACTOME_INTERFERON_SIGNALING                         2.0           True
REACTOME_INTERFERON_ALPHA_BETA_SIGNALING              2.0           True
REACTOME_CYTOKINE_SIGNALING_IN_IMMUNE_SYSTEM          2.0           True
REACTOME_ANTIVIRAL_MECHANISM_BY_IFN_STIMULATED_...    2.0           True
REACTOME_RIG_I_MDA5_MEDIATED_INDUCTION_OF_IFN_A...    2.0           True

                                                              comparison  \
REACTOME_INTERFERON_SIGNALING                         stimulated vs. rest
REACTOME_INTERFERON_ALPHA_BETA_SIGNALING              stimulated vs. rest
REACTOME_CYTOKINE_SIGNALING_IN_IMMUNE_SYSTEM          stimulated vs. rest
REACTOME_ANTIVIRAL_MECHANISM_BY_IFN_STIMULATED_...    stimulated vs. rest
REACTOME_RIG_I_MDA5_MEDIATED_INDUCTION_OF_IFN_A...    stimulated vs. rest

                                                        group1 group2
REACTOME_INTERFERON_SIGNALING                         stimulated   rest
REACTOME_INTERFERON_ALPHA_BETA_SIGNALING              stimulated   rest
REACTOME_CYTOKINE_SIGNALING_IN_IMMUNE_SYSTEM          stimulated   rest
REACTOME_ANTIVIRAL_MECHANISM_BY_IFN_STIMULATED_...    stimulated   rest
REACTOME_RIG_I_MDA5_MEDIATED_INDUCTION_OF_IFN_A...    stimulated   rest
```

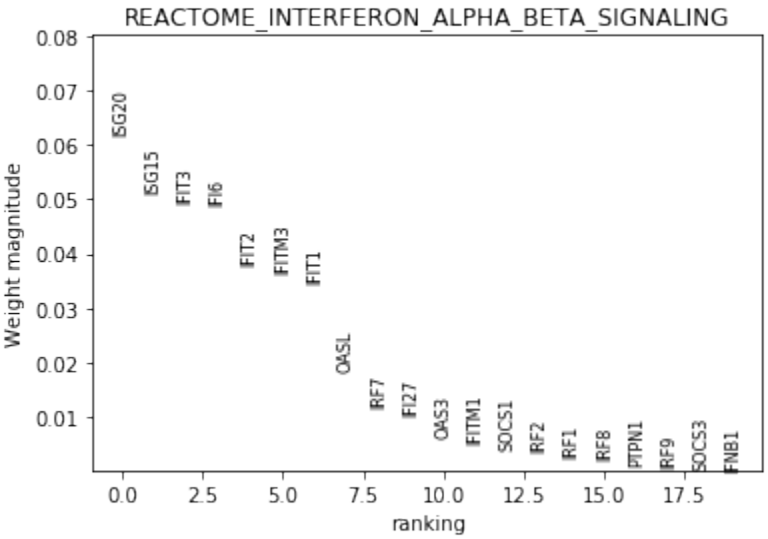
As expected, most of the top hits in the stimulated popultation are related to the activation of interferon pathways and immune signaling. We can display the results in an analog of a volcano plot using bayes factors to encode significance.

[13]:
```
vega.plotting.volcano(model.adata, group1='stimulated', group2='rest', title='Stimulated␣
↪vs. Rest')
```



Additionally, we can visualize the weights attributed to each gene in a given pathway to understand the main drivers of variability in the dataset.

[14]:
```
vega.plotting.rank_gene_weights(model, gmv_list=['REACTOME_INTERFERON_ALPHA_BETA_
↪SIGNALING'], n_genes=20, color_in_set=False)
```



Exporting results

We can use pandas to export the differential activity results to look at it later.

[15]:
```
#da_df.to_csv('./da_results.tsv', sep='\t')
```

3.4 References

BIBLIOGRAPHY

[Sokolov2016] Artem Sokolov, Daniel E. Carlin, Evan O. Paull, Robert Baertsch, Joshua M. Stuart (2016), *Pathway-Based Genomics Prediction using Generalized Elastic Net*, PLOS Computational Biology.

[Seninge2021] Lucas Seninge, Ioannis Anastopoulos, Hongxu Ding, Joshua Stuart (2021), *VEGA is an interpretable generative model for inferring biological network activity in single-cell transcriptomics*, Nature Communications.

[Gayoso2022] Adam Gayoso*, Romain Lopez*, Galen Xing*, Pierre Boyeau, Valeh Valiollah Pour Amiri, Justin Hong, Katherine Wu, Michael Jayasuriya, Edouard Mehlman, Maxime Langevin, Yining Liu, Jules Samaran, Gabriel Misrachi, Achille Nazaret, Oscar Clivio, Chenling Xu, Tal Ashuach, Mohammad Lotfollahi, Valentine Svensson, Eduardo da Veiga Beltrame, Vitalii Kleshchevnikov, Carlos Talavera-Lopez, Lior Pachter, Fabian J Theis, Aaron Streets, Michael I Jordan, Jeffrey Regier, and Nir Yosef (2022), *A Python library for probabilistic analysis of single-cell omics data*, Nature Biotechnology.

[Lopez2018] Romain Lopez, Jeffrey Regier, Michael Cole, Michael I. Jordan, Nir Yosef (2018), *Deep generative modeling for single-cell transcriptomics*, Nature Methods.

[Rybakov2020] Sergei Rybakov, Mohammad Lotfollahi, Fabian J. Theis, F. Alexander Wolf (2021), *Learning interpretable latent autoencoder representations with annotations of feature sets*, biorxiv.

[Boyeau2019] Pierre Boyeau, Romain Lopez, Jeffrey Regier, Adam Gayoso, Michael I. Jordan, Nir Yosef (2019), *Deep generative models for detecting differential expression in single cells*, Machine Learning in Computational Biology (MLCB).

PYTHON MODULE INDEX

V

INDEX