epi

Release 0.1

Sean Bittner

Jun 08, 2021

CONTENTS:

This package provides EPI for general python implementations of neural circuit models based on the algorithm in https://www.biorxiv.org/content/10.1101/837567v2.abstract.

CHAPTER

ONE

INSTALLATION

1. Download the epi package from github.:

```
git clone https://github.com/cunningham-lab/epi.git
```

2. Install epi package and dependencies.:

```
cd epi/
pip install .
```

In order to create videos visualizing the optimization, you'll need to install ffmpeg.

`https://ffmpeg.org/documentation.html<https://ffmpeg.org/documentation.html>`_

CHAPTER

TWO

TUTORIAL

For a tutorial, check out the jupyter notebook for oscillating 2D linear dynamical systems.

1. Getting Started with EPI
2. Choosing a normalizing flow architecture
3. Choosing hyperparameters for augmented Lagrangian optimization

CHAPTER

THREE

API

3.1 epi.models

Models.

class epi.models.**Distribution**(*nf*, *parameters=None*)

Bases: object

Distribution class with numpy UI, and tensorflow-enabled methods.

Obtain samples, log densities, gradients and Hessians of a distribution defined by a normalizing flow optimized via tensorflow.

Parameters

- **nf** (*epi.normalizing_flows.NormalizingFlow*) – Normalizing flow trained via tensorflow.
- **parameters** (*list, optional*) – List of *epi.models.Parameter*. Defaults to z1, ..

gradient(*z*)

Calculates the gradient $\nabla_z \log p(z))$.

Parameters **z** (*np.ndarray*) – Parameter vector.

Returns Gradient of log probability with respect to z.

Return type np.ndarray

hessian(*z*)

Calculates the Hessian $\frac{\partial^2 \log p(z)}{\partial z \partial z^\top}$.

Parameters **z** (*np.ndarray*) – Parameter vector.

Returns Hessian of log probability with respect to z.

Return type np.ndarray

log_prob(*z*)

Calculates log probability of samples from distribution.

Parameters **z** (*np.ndarray*) – Parameter vector.

Returns Log probability of samples.

Return type np.ndarray

plot_dist(*z*, *c=None*, *kde=True*)

Generates pairplot of parameters.

Parameters

- **z** (*np.ndarray*) – Parameter vectors (N, D).
- **c** (*np.ndarray*) – Color value for parameters. (Defaults to log probability).
- **kde** (*bool*) – If True, bottom-left plots shows kde contours of c.

Returns Hessian of log probability with respect to z.

Return type np.ndarray

sample(*N*)
Sample N times.

Parameters **N** (*int:*) – Number of samples.

Returns N samples.

Return type np.ndarray

set_batch_norm_trainable(*trainable*)

class epi.models.**Model**(*name*, *parameters*)
Bases: object

Model to run emergent property inference on. To run EPI on a model:

1. Initialize an *epi.models.Model* with a list of *epi.models.Parameter*.
2. Use *epi.models.Model.set_eps* to set the emergent property statistics of the model.
3. Run emergent property inference for mean parameter μ using *epi.models.Model.epi*.

Parameters

- **name** (*str*) – Name of model.
- **parameters** (*list*) – List of *epi.models.Parameter*.

epi(*mu*, *arch_type='coupling'*, *num_stages=3*, *num_layers=2*, *num_units=50*, *elemwise_fn='affine'*, *batch_norm=False*, *bn_momentum=0.0*, *post_affine=True*, *random_seed=1*, *init_type=None*, *init_params=None*, *K=10*, *num_iters=1000*, *N=500*, *lr=0.001*, *c0=1.0*, *gamma=0.25*, *beta=4.0*, *alpha=0.05*, *nu=1.0*, *stop_early=False*, *log_rate=50*, *verbose=False*, *save_movie_data=False*)
Runs emergent property inference for this model with mean parameter μ.

Parameters

- **mu** (*np.ndarray*) – Mean parameter of the emergent property.
- **arch_type** (*str, optional*) – ∈ ['autoregressive', 'coupling'], defaults to 'coupling'.
- **num_stages** (*int, optional*) – Number of coupling or autoregressive stages, defaults to 3.
- **num_layers** (*int, optional*) – Number of neural network layer per conditional, defaults to 2.
- **num_units** (*int, optional*) – Number of units per layer, defaults to max(2D, 15).
- **elemwise_fn** (*str, optional*) – Inter-stage bijector *in* ['affine', 'spline'], defaults to 'affine'.
- **batch_norm** (*bool, optional*) – Use batch normalization between stages, defaults to True.

- **bn_momentum** – Batch normalization momentum parameter, defaults to 0.99.
- **post_affine** (*bool, optional*) – Shift and scale following main transform, defaults to False.
- **random_seed** (*int, optional*) – Random seed of architecture parameters, defaults to 1.
- **init_type** (*str, optional*) – ∈ ['gaussian', 'abc'].
- **init_params** (*dict, optional*) – Parameters according to init_type.
- **K** (*int, float, optional*) – Number of augmented Lagrangian iterations, defaults to 10.
- **num_iters** (*int, optional*) – Number of optimization iterations, defaults to 1000.
- **N** (*int, optional*) – Number of batch samples per iteration, defaults to 500.
- **lr** (*float, optional*) – Adam optimizer learning rate, defaults to 1e-3.
- **c0** (*float, optional*) – Initial augmented Lagrangian coefficient, defaults to 1.0.
- **gamma** (*float, optional*) – Augmented lagrangian hyperparameter, defaults to 0.25.
- **beta** (*float, optional*) – Augmented lagrangian hyperparameter, defaults to 4.0.
- **alpha** (*float, optional*) – P-value threshold for convergence testing, defaults to 0.05.
- **nu** (*float, optional*) – Fraction of N for convergence testing, defaults to 0.1.
- **stop_early** (*bool, optional*) – Exit if converged, defaults to False.
- **log_rate** (*int, optional*) – Record optimization data every so iterations, defaults to 100.
- **verbose** (*bool, optional*) – Print optimization information, defaults to False.
- **save_movie_data** (*bool, optional*) – Save data for making optimization movie, defaults to False.

Returns q_theta, opt_df, save_path, failed

Return type *epi.models.Distribution*, pandas.DataFrame, str, bool

epi_opt_movie(*path*)
Generate video of EPI optimization.

Parameters **path** – Path to folder with optimization data.

get_convergence_epoch(*init_params*, *nf*, *mu*, *aug_lag_hps*, *alpha=0.05*, *nu=1.0*, *mu_test=None*, *start_k=0*, *sampling_seed=0*)

get_epi_df()

get_epi_dist(*df_row*, *k=None*)

get_epi_path(*init_params*, *nf*, *mu*, *AL_hps*, *eps_name=None*)

parameter_check(*parameters*, *verbose=False*)
Check that model parameter list has no duplicates and valid bounds.

Parameters

- **parameters** (*list*) – List of *epi.models.Parameter*.

- **verbose** (*bool, optional*) – Print rationale for check failure if True, defaults to False.

Returns True if parameter list is valid.

Return type bool

set_eps(*eps*)
Set the emergent property statistic calculation for this model.

The arguments of eps should be batch vectors of univariate parameter tensors following the naming convention in `self.Parameters`.

Parameters **eps** (*function*) – Emergent property statistics function.

test_convergence(*R_means*, *alpha*, *verbose=False*, *ind=None*)
Tests convergence of EPI constraints.

Parameters

- **R_means** (*np.ndarray*) – Emergent property statistic means.
- **alpha** (*float*) – P-value threshold.

class epi.models.**Parameter**(*name*, *D*, *lb=None*, *ub=None*)
Bases: object

Univariate parameter of a model.

Parameters

- **name** (*str*) – Parameter name.
- **D** (*int*) – Number of dimensions of parameter.
- **lb** (*np.ndarray, optional*) – Lower bound of variable, defaults to *np.NINF*np.ones(D)*.
- **ub** (*np.ndarray, optional*) – Upper bound of variable, defaults to *np.PINF*np.ones(D)*.

epi.models.**format_opt_msg**(*k*, *i*, *cost*, *H*, *R*, *time_per_it*)

epi.models.**get_R_mean_dist**(*nf*, *eps*, *mu*, *M*, *N*)

epi.models.**get_R_norm_dist**(*nf*, *eps*, *mu*, *M*, *N*)

epi.models.**two_dim_T_x_batch**(*nf*, *eps*, *M*, *N*, *m*)

3.2 epi.normalizing_flows

Normalizing flow architecture class definitions for param distributions.

class epi.normalizing_flows.**IntervalFlow**(*lb*, *ub*)
Bases: tensorflow_probability.bijectors.Bijector

Bijector maps from $\mathcal{R}^N$ to an interval.

Each dimension is handled independently according to the type of bound.

- no bound: $y_i = x_i$
- only lower bound: $y_i = \log(1 + \exp(x_i)) + lb_i$
- only upper bound: $y_i = -\log(1 + \exp(x_i)) + ub_i$

- upper and lower bound: $y_i = (ub_i - lb_i)(x_i) + \frac{ub_i + lb_i}{2}$

Parameters

- **lb** (*np.ndarray*) – Lower bound. N values are numeric including `float('-inf')`.
- **ub** (*np.ndarray*) – Upper bound. N values are numeric including `float('inf')`.

forward(*x*)

Runs bijector forward and calculates log det jac of the function.

Parameters **x** (*tf.Tensor*) – Input tensor.

Returns The forward pass of the interval flow

Return type (tf.Tensor, tf.Tensor)

forward_and_log_det_jacobian(*x*)

Runs bijector forward and calculates log det jac of the function.

It's more efficient to run samples and ldjs forward together for EPI.

Parameters **x** (*tf.Tensor*) – Input tensor.

Returns The forward pass and log determinant of the jacobian.

Return type (tf.Tensor, tf.Tensor)

forward_log_det_jacobian(*x*)

Calculates forward log det jac of the interval flow.

Parameters **x** (*tf.Tensor*) – Input tensor.

Returns Log determinant of the jacobian of interval flow.

Return type (tf.Tensor, tf.Tensor)

inverse(*x*)

Inverts bijector at value x.

Parameters **x** (*tf.Tensor*) – Input tensor.

Returns The backward pass of the interval flow

Return type (tf.Tensor, tf.Tensor)

inverse_log_det_jacobian(*x*, *event_ndims=1*)

Log determinant jacobian of inverse pass.

Parameters **x** (*tf.Tensor*) – Input tensor.

Returns The inverse log determinant jacobian.

Return type (tf.Tensor, tf.Tensor)

class epi.normalizing_flows.**NormalizingFlow**(*arch_type*, *D*, *num_stages*, *num_layers*, *num_units*, *elemwise_fn='affine'*, *num_bins=4*, *batch_norm=True*, *bn_momentum=0.0*, *post_affine=True*, *bounds=None*, *random_seed=1*)

Bases: `tensorflow.keras.Model`

Normalizing flow network for approximating parameter distributions.

The normalizing flow is constructed via stage(s) of either coupling or autoregressive transforms of q_0. Coupling transforms are real NVP bijectors where each conditional distribution has the same number of neural network layers and units. One stage is one coupling (second half of elements are conditioned on the first half (see

`tfp.bijectors.RealNVP`)). Similarly, autoregressive transforms are masked autoregressive flow (MAF) bijectors. One stage is one full autoregressive factorization (see `tfp.bijectors.MAF`).

After each stage, which is succeeded by another coupling or autoregressive transform, the dimensions are permuted via a Glow permutation. This facilitates randomized conditioning (real NVP) and factorization orderings (MAF) at each stage.

E.g. `arch_type='autoregressive', num_stages=2`

q_0 -> MAF -> permute -> MAF -> ...

We parameterize the final processing stages of the normalizing flow (a deep generative model) via post_affine and bounds.

To facilitate scaling and shifting of the normalizing flow up to this point, one can set post_affine to True.

E.g. `arch_type='autoregressive', num_stages=2, post_affine=True`

q_0 -> MAF -> permute -> batch norm -> MAF -> affine -> ...

By setting bounds to a tuple (lower_bound, upper_bound), the final step in the normalizing flow maps to the support of the distribution using an *epi.normalizing_flows.IntervalFlow*.

E.g. `arch_type='autoregressive', num_stages=2, post_affine=True, bounds=(lb,ub)`

q_0 -> MAF -> permute -> batch norm -> MAF -> post affine -> interval flow

The base distribution q_0 is chosen to be a standard isotoropic gaussian. Transforms of coupling and autoregressive layers can be parameterized as an 'affine' function or a 'spline' via parameter elemwise_fn.

Parameters

- **arch_type** (*str*) – ∈ *['autoregressive', 'coupling']*
- **D** (*int*) – Dimensionality of the normalizing flow.
- **num_stages** (*int*) – Number of coupling or autoregressive stages.
- **num_layers** (*int*) – Number of neural network layer per conditional.
- **num_units** (*int*) – Number of units per layer.
- **elemwise_fn** (*str, optional*) – Inter-stage bijector *in* `['affine', 'spline']`, defaults to 'affine'.
- **num_bins** (*int, optional*) – Number of bins when elemwise_fn is spline.
- **batch_norm** (*bool, optional*) – Use batch normalization between stages, defaults to True.
- **bn_momentum** – Batch normalization momentum parameter, defaults to 0.99.
- **post_affine** (*bool, optional*) – Shift and scale following main transform.
- **bounds** (*(np.ndarray, np.ndarray), optional*) – Bounds of distribution support, defaults to None.
- **random_seed** (*int, optional*) – Random seed of architecture parameters, defaults to 1.

gauss_KL(*z*, *log_q_z*, *mu*, *Sigma*)

get_init_path(*mu*, *Sigma*)

initialize(*mu*, *Sigma*, *N=500*, *num_iters=10000*, *lr=0.001*, *log_rate=100*, *load_if_cached=True*, *save=True*, *verbose=False*)

Initializes architecture to gaussian distribution via variational inference.

$$\underset{q_\theta \in Q}{\operatorname{argmax}} H(q_\theta) + \eta^\top \mathbb{E}_{z \sim q_\theta}[T(z)]$$

where η and $T(z)$ for a multivariate gaussian are:

$$\eta = \begin{bmatrix} \Sigma^{-1}\mu \\ \operatorname{vec}\left(-\frac{1}{2}\Sigma^{-1}\right) \end{bmatrix} \quad T(z) = \begin{bmatrix} z \\ \operatorname{vec}\left(zz^\top\right) \end{bmatrix}$$

Parameter *init_type* may be:

`'iso_gauss'` with parameters

- `init_params.loc` set to scalar mean of each variable.
- `init_params.scale` set to scale of each variable.

`'gaussian'` with parameters

- `init_params.mu` set to the mean.
- `init_params.Sigma` set to the covariance.

Parameters

- **init_type** (*str*) – ∈ *['iso_gauss', 'gaussian']*
- **init_params** (*dict*) – Parameters according to `init_type`.
- **N** (*int*) – Number of batch samples per iteration.
- **num_iters** (*int, optional*) – Number of optimization iterations, defaults to 500.
- **lr** (*float, optional*) – Adam optimizer learning rate, defaults to 1e-3.
- **log_rate** (*int, optional*) – Record optimization data every so iterations, defaults to 100.
- **load_if_cached** (*bool, optional*) – If initialization has been optimized before, load it, defaults to True.
- **save** (*bool, optional*) – Save initialization if true, defaults to True.
- **verbose** (*bool, optional*) – Print verbose output, defaults to False.

sample(*N*)

Generate N samples from the network.

Parameters **N** (*int*) – Number of samples.

Returns N samples and log determinant of the jacobians.

Return type (tf.Tensor, tf.Tensor)

to_string()

Converts architecture to string for file saving.

Returns A unique string for the architecture parameterization.

Return type str

class epi.normalizing_flows.**SplineParams**(*D*, *num_layers*, *num_units*, *nbins=32*, *B=1*)

Bases: `tensorflow.Module`

epi.normalizing_flows.**hp_df_to_nf**(*hp_df*, *model*)

epi.normalizing_flows.**trainable_lu_factorization**(*event_size*, *batch_shape=()*, *seed=None*, *dtype=tensorflow.float32*, *name=None*)

3.3 epi.util

General util functions for EPI.

class epi.util.**AugLagHPs**(*N=1000*, *lr=0.001*, *c0=1.0*, *gamma=0.25*, *beta=4.0*)
Bases: object

Augmented Lagrangian optimization hyperparamters.

Parameters

- **N** (*int, optional*) – Batch size, defaults to 1000.
- **lr** (*float, optional*) – Learning rate, defaults to 1e-3.
- **c0** (*float, optional*) – L-2 norm on R coefficient, defaults to 1.0.
- **gamma** (*float, optional*) – Epoch reduction factor for epoch, defaults to 1/4.
- **beta** (*float, optional*) – L-2 norm magnitude increase factor.

to_string()
String for filename involving hyperparameter setting.

Returns Hyperparameters as a string.

Return type str

epi.util.**array_str**(*a*)
Returns a compressed string from a 1-D numpy array.

Parameters **a** (*str*) – A 1-D numpy array.

Returns A string compressed via scientific notation and repeated elements.

Return type str

epi.util.**aug_lag_vars**(*z*, *log_q_z*, *eps*, *mu*, *N*)
Calculate augmented lagrangian variables requiring gradient tape.

$H(\theta) = \mathbb{E}_{z\sim q_\theta}[-\log(q_\theta(z)]$

$R(\theta) = \mathbb{E}_{z\sim q_\theta, x\sim p(x|z)}[T(x) - \mu]$

$R_1(\theta) = \mathbb{E}_{z_1\sim q_\theta, x\sim p(x|z_1)}[T(x) - \mu]$

$R_2(\theta) = \mathbb{E}_{z_2\sim q_\theta, x\sim p(x|z_2)}[T(x) - \mu]$

where θ are params and z_1, z_2 are the two halves of the batch samples.

Parameters

- **z** (*tf.Tensor*) – Parameter samples.
- **log_q_z** (*tf.Tensor*) – Parameter sample log density.
- **eps** (*function*) – Emergent property statistics function.
- **mu** (*np.ndarray*) – Mean parameter of the emergent property.
- **N** (*int*) – Number of batch samples.

Returns $H(\theta)$, $R(\theta)$, list $R_1(\theta)$ by dimension, and $R_2(\theta)$.

Return type list

epi.util.**check_bound_param**(*bounds*, *param_name*)

epi.util.**dbg_check**(*tensor*, *name*)

epi.util.**filter_outliers**(*c*, *num_stds=4*)

epi.util.**gaussian_backward_mapping**(*mu*, *Sigma*)
Calculates natural parameter of multivaraite gaussian from mean and cov.

Parameters

- **mu** (*np.ndarray*) – Mean of gaussian
- **Sigma** (*np.ndarray*) – Covariance of gaussian.

Returns Natural parameter of gaussian.

Return type np.ndarray

epi.util.**get_conditional_mode**(*dist*, *ind*, *val*, *z0=None*, *lr=1e-06*, *num_steps=100*, *decay=1.0*, *decay_steps=100*)

epi.util.**get_dir_index**(*path*)

epi.util.**get_hash**(*hash_vars*)

epi.util.**get_max_H_dist**(*model*, *epi_df*, *mu*, *alpha=0.05*, *nu=1.0*, *check_last_k=None*, *by_df=False*)

epi.util.**np_column_vec**(*x*)
Takes numpy vector and orients it as a n x 1 column vec.

Parameters **x** (*np.ndarray*) – Vector of length n

Returns n x 1 numpy column vector

Return type np.ndarray

epi.util.**pairplot**(*Z*, *dims*, *labels*, *lb=None*, *ub=None*, *clims=None*, *ticks=None*, *c=None*, *c_label=None*, *cmap=None*, *s=50*, *s_star=100*, *starred=None*, *c_starred=None*, *star_marker='*'*, *traj=None*, *fontsize=12*, *figsize=(12, 12)*, *outlier_stds=10*, *ticksize=None*, *labelpads=None*, *unity_line=False*, *subplots=None*, *skip_cbar=False*, *pfname='images/temp.png'*)

epi.util.**plot_T_x**(*T_x*, *T_x_sim*, *bins=30*, *xmin=None*, *xmax=None*, *x_mean=None*, *x_std=None*, *figsize=None*, *xlabel=None*, *ylim=None*, *fontsize=14*)

epi.util.**plot_opt**(*epi_df*, *max_k=None*, *cs=None*, *fontsize=12*, *H_ylim=None*, *figdir='./'*, *save=False*)

epi.util.**plot_square_mat**(*ax*, *A*, *c='k'*, *lw=4*, *fontsize=12*, *bfrac=0.05*, *title=None*, *xlims=None*, *ylims=None*, *text_c='k'*)

epi.util.**sample_aug_lag_hps**(*n, N_bounds=[200, 1000], lr_bounds=[0.0001, 0.01], c0_bounds=[0.001, 1000.0], gamma_bounds=[0.1, 0.5]*)
Samples augmented Lagrangian parameters from uniform distribution.

Parameters **N_bounds** – Bounds on batch size.

epi.util.**set_dir_index**(*index*, *index_file*)

epi.util.**unbiased_aug_grad**(*R1s*, *R2*, *params*, *tape*)
Unbiased gradient of the l-2 norm of stochastic constraint violations.

$R_1(\theta) = \mathbb{E}_{z_1 \sim q_\theta, x \sim p(x|z_1)}[T(x) - \mu]$

$R_2(\theta) = \mathbb{E}_{z_2 \sim q_\theta, x \sim p(x|z_2)}[T(x) - \mu]$

where θ are params and z_1, z_2 are the two halves of the batch samples.

The augmented gradient is computed as

$\nabla_\theta ||R(\theta)||^2 = 2\nabla_\theta R_1(\theta) \cdot R_2(\theta)$

Parameters

- **R1s** (*list*) – Mean constraint violation over first half of samples.
- **R2** (*tf.Tensor*) – Mean constraint violation over the second half of samples.
- **params** (*list*) – Trainable variables of q_θ
- **tape** (*tf.GradientTape*) – Persistent gradient tape watching params.

Returns Unbiased gradient of augmented term.

Return type list

3.4 epi.error_formatters

Format strings for epi package errors.

epi.error_formatters.**format_type_err_msg**(*obj*, *arg_name: str*, *arg*, *correct_type*) → str
Formats error message for incorrect types.

Parameters

- **obj** (*object*) – The function or class to which argument was supplied.
- **arg_name** (*str*) – Name of function argument.
- **arg** – The argument with incorrect type.
- **correct_type** – The correct type of the argument.

Returns Type error message.

Return type str

3.5 epi.example_eps

Example emergent property statistics.

epi.example_eps.**linear2D_freq**(*a11*, *a12*, *a21*, *a22*)
Linear 2D system frequency response characterisitcs.

For a two-dimensional linear system:

$\tau \dot{x} = Ax$

$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$

We can characterize the dynamics using the real and imaginary components of the primary eigenvalue λ_1, of C that which has the greates real component or secondarily the greatest imaginary component if the two eigenvalues are equal, where $C = A/\tau$.

$$T(x) = \begin{bmatrix} \text{real}(\lambda_1) \\ \text{real}(\lambda - 0)^2 \\ 2\pi\text{imag}(\lambda_1) \\ (\text{imag}(\lambda_1) - 2\pi)^2 \end{bmatrix}$$

Parameters

- **a11** (*tf.Tensor*) – Dynamics coefficient.
- **a12** (*tf.Tensor*) – Dynamics coefficient.
- **a21** (*tf.Tensor*) – Dynamics coefficient.
- **a22** (*tf.Tensor*) – Dynamics coefficient.

epi.example_eps.**linear2D_freq_np**(*a11*, *a12*, *a21*, *a22*)

epi.example_eps.**linear2D_freq_sq**(*A*)

CHAPTER

FOUR

INDICES AND TABLES

- genindex

PYTHON MODULE INDEX

e

INDEX

P

S

T

U