**Organization of neural\_imputation**

*(tailored to importance weighted autoencoder)*

**neural\_imputation/**

│

├── **main.py** - main script to start training, testing, and imputing

│

├── parse\_config.py - class to parse model config json and cli options

│

├── trainer.py - class to run the train/val/testing phases

│

├── base/ - abstract base classes

│ ├── base\_data\_loader.py

│ ├── base\_model.py

│ └── base\_trainer.py

│

├── data\_loader/ - anything about data loading goes here

│ ├── datasets.py - class for handling datasets with missing masks

│ └── **preprocessor.py** – class for preprocessing data and preimputing

│

├── model/ - architecture for the neural network models

│ ├── autoencoder.py – generic classes for Encoder and Decoder

│ ├── adversarial.py – generic classes for Generator and Discriminator

│ ├── loss.py – loss functions for training the models

│ ├── metrics.py – other metric functions for evaluating models

│ └── **models.py** – definition of major model classes (VAE/IWAE/GANs etc) │

├── utils/ - utility functions for use throughout

│ ├── **distribution.py** – Distribution wrapper with added functionality

│ ├── **metrics.py** – metrics functions for evaluating models

│ ├── logger.py – class for logging output from model

│ └── util.py – various other utility functions

**Basic pipeline:**

1. Generate data and masks using R script and save as CSV
2. Create the neural\_imputation configuration JSON defining:
   1. Preprocessing to data
      1. Treat data as numeric or discrete
      2. Standardization or normalization
      3. Noise injection and type of noise
   2. Initial preimputation for “missing” value (zero, noise, mean etc)
   3. Model architecture
   4. Metrics logged
3. Run main.py script

**JSON configuration**

﻿{"name": "f5v50n1000\_corr",

"miss\_spike\_percent": 0.2,

"batch\_size": 100,

"arch": {"type": "VAE",

"args": {"q\_distribution": "laplace",

"p\_distribution", "studentt",

"z\_prior": "laplace",

"input\_size": 50,

"code\_size": 50,

"encoder\_hidden\_sizes"=[10],

"decoder\_hidden\_sizes"=[10],

"K\_train": 10,

"K\_test": 100}},

"dataset": {"type": "MaskedDataset"},

"preprocessor": {"type": "Preprocessor",

"args":{"dtype": "numeric",

"scaler": "StandardScaler",

"noise": True,

"noise\_level": 0.5,

"test\_split": 0.1,

"validation\_split": 0.2,

"preimputation": "zero"}},

"optimizer": {"type": "Adam",

"args":{"lr": 0.001,

"weight\_decay": 0,

"amsgrad": true}},

"loss": "iwae\_loss",

"metrics": ["mse\_true", "riemannian", "frobenius"],

"lr\_scheduler": {"type": "StepLR",

"args": {"step\_size": 50,

"gamma": 0.1}},

"trainer": {"epochs": 300,

"imputation": {"type": "single",

"mode": "samples"}

"save\_dir": "saved/",

"save\_period": 1,

"verbosity": 2,

"monitor": "min val\_loss",

"early\_stop": 10,

"tensorboard": true}

}

**Rate–distortion theory**

As mentioned earlier, autoencoders can be thought of as generalized information encoder, with the latent space functioning as the channel.