User Guide for Electron-Passing Graph Neural Network

This user guide helps using the Electron Passing Graph Neural Network (EPGNN) from the interdisciplinary project of *Magnus Wagner* in the M.Sc. Data Science studies @ TU Vienna. The preprocessing pipeline and following model help predicting atom charges in ionic structures like salts or ionic liquids.

1. **Create a suitable dataset.**

   Apart from the work done in this project, the training data needs to be preprocessed and calculated in the correct format. As input data, the following information is known for each atom in a sample system:

   a) Two-dimensional Bessel descriptors containing information about the atomic number and the neighborhood of the atom
   b) Charge calculation processed by DFT approach. (Used as ground truth)
   c) Relative position to a fixed point to able to calculate distances between different atoms
   d) Atomic number

   To be able to calculate the distances, it is additionally necessary to know about geometry and size of the unit cell.

2. **Adapt the preprocessing pipeline if necessary**

   Preprocessing is currently supported for the flat crystal structures of SrTiO3(110) with periodic boundary conditions in two dimensions and the ionic liquid ethylammonium nitrate (EAN) with periodic boundary conditions in all three dimensions. Both structures have orthogonal unit cells. If your chemical structure has a different unit cell geometry or different periodic boundary conditions, you will need to adapt the distance calculations between different atoms to be represented correctly. To do this, the function *center_at_atoms_diagonal()* in the file *preprocessing_base.py* needs to be adapted.

3. **Add necessary information in the file presets.json**

   The file *presets.json* should contain the chemical formula of the compound as a key with another dictionary as its value. This one should contain the *path* to the database, a *symbol_map* which is mapping the element symbols to an index and a *charge_map* which is mapping the indices (as strings) to their average charge. Additionally, it needs to contain the *total_charge* of the system. The average charge can be approximate but when adding up all average charges by multiplying them with the cardinality from the chemical formula, it needs to sum up to the total charge of the system. A last variable is the *color_sequence* for visualization the different atoms. You will need to experiment a bit with the correct order of the colors to get your visualization the way you want it.

4. Run the **hyperparameter optimization pipeline** with different settings to test which settings are suitable for predicting atomic charges in your chemical structure. The following inputs can be used: Change the DEFAULT_DICT to change the default parameters for the model.

```
"""Input:
    - DEFAULT_DICT: Dictionary with all default values in it for the following parameters
    - E_DIM [int]: Number of embedding dimensions for the edges
    - R_SWITCH [float]: First parameter of the cutoff function. A lower value leads to a less steep
      slope.
    - R_CUT [float]: Second parameter of the cutoff function. The cutoff function approaches zero
      close to this value.
    - DISTANCE_ENCODING_TYPE ["none", "root", "log", "gaussian"]: You can encode the distances between
      atoms with the following distance encoding types before they are embedded.
          o "none": no distance encoding before embedding the distances
          o "root": square root of the distances before embedding the distances.
          o "log1": logarithmic value of (distance+1) [to avoid negative values]
```

```
        -   FEATURES [array]: An array of dimensions for the underlying neural network. It needs to end with
            1, but you can increase the complexity to improve model performance. e.g. [128,64,32,16,8,1]
        -   NUM_PASSES [int]: number of passing steps for the message-passing algorithm applied to the graph.
        -   ACTIVATION [“relu”,”switch”] Two different activation functions to choose from.
        -   N_EPOCHS [int]: The number of training epochs the model trains.
        -   LR [float]: The initial learning rate for the AdamW-optimizer. It will have an effect on the
            weight decay (which is proportional to the learning rate in the library.)
        -   WEIGHT_DECAY [float]: The weight decay after each epoch to avoid overfitting.
        -   OPTIM_DICT: Dictionary with arrays of input parameters to run hyperparameter optimization
        -   OVERWRITE: [bool] -> Flag to decide if the model should run a pipeline again that is already
            saved in the results. (False means no overwriting of previous results)
        -   FORMULA: [str] Formula of the chemical compound

    Output: None (but results are saved in result list in /results)
    """
```

The settings for the hyperparameter optimization pipeline can be set in the variable OPTIM_DICT. Just set an array of options for the specific key and the pipeline will run through all of them. Watch out of setting arrays for two hyperparameters at once as the pipeline will run through every combination of them! The hyperparameter optimization pipeline tracks the validation set RMSEs for every 10 epochs. The **results** are typically saved in the file *results/{formula}.csv* with their setting as the index. They contain the hyperparameter setting, the time taken for the training run, the test RMSE, the test MAE and the best validation RMSE.

5. **Train the model with the best hyperparameter configuration** by setting the necessary parameters and running the function *train_single_model().* You can set SAVE_MODEL to True to save the model with the best validation RMSE from the training run in the folder *models.*

```
Input:
 -    E_DIM [int]: Number of embedding dimensions for the edges
 -    R_SWITCH [float]: First parameter of the cutoff function. A lower value leads to a less steep slope.
 -    R_CUT [float]: Second parameter of the cutoff function. The cutoff function approaches zero close to this
      value.
 -    DISTANCE_ENCODING_TYPE [“none”, “root”, “log”, “gaussian”]: You can encode the distances between atoms
      with the following distance encoding types before they are embedded.
            -    “none”: no distance encoding before embedding the distances
            -    “root”: square root of the distances before embedding the distances.
            -    “log1”: logarithmic value of (distance+1) [to avoid negative values]
 -    FEATURES [array]: An array of dimensions for the underlying neural network. It needs to end with 1, but
      you can increase the complexity to improve model performance e.g. [128,64,32,16,8,1]
 -    NUM_PASSES [int]: number of passing steps for the message-passing algorithm applied to the graph.
 -    ACTIVATION [“relu”,”switch”] Two different activation functions to choose from.
 -    N_EPOCHS [int]: The number of training epochs the model trains.
 -    OVERWRITE: [bool] -> Flag to decide if the model should run a pipeline again that is already saved in
      the results. (False means no overwriting of previous results)
 -    FORMULA: [str] Formula of the chemical compound
 -    SAVE_MODEL: [bool] -> Should the model be saved after training or not?
 -    SAMPLE_SIZE: [int] -> How many samples from the database shall be used for training, "None" equals all
      data.
 -    LR [float]: The initial learning rate for the AdamW-optimizer. It will have an effect on the weight decay
      (which is proportional to the learning rate in the library.)
 -    WEIGHT_DECAY [float]: The weight decay after each epoch to avoid overfitting.
```

6. **Visualize results with the function *visualize_results()*.** Visualization plots ground truth charges against predicted charges.

```
""" Visualize the results of a model on a batch of data about a chemical structure.
Input:
      o    model_path: str -> Path to the saved model.
      o    FORMULA: str -> Formula of the chemical compound.
      o    batches [optional]: dict -> Dictionary from the training pipeline train_single_model()
      o    db_path [optional]: str -> Path to data to infer. Should be similar to data the model was trained
           on!
      o    xrange [optional]: (float, float) -> Range to plot for x- and y-axis
      o    save_name [optional]: str -> If not None -> Name to folder to save the plot to.
Output:
      o    Image of plot.
"""
```

7. **Run inference with the function *infer().*** Inference is batched so you do not have to worry about a dataset that is too large. With max_batch_size you can decrease the batch-size to a number that works with your VRAM.

```
""" Run inference on a new dataset.
Input:
 -    model_path: str -> Path to the saved model.
```

```
    -    db_path [optional]: str -> Path to data to infer. Should be similar to data the model was trained on!
    -    formula: str -> Formula of the chemical compound.
    -    max_batch_size: int -> Maximum number of samples per batch

    Output:

    -    Tensor of predicted samples with dimension (n_samples, n_atom)
    """
```

8. **Experiment with the notebook.ipynb for setting up different functions.**
   The notebook was used to run hyperparameter optimization and has the necessary variables already setup for running pipelines, inference and visualization