i-Melt

Release 2.0.0

Charles Le Losq, Barbara Baldoni, Andrew P. Valentine

Jul 05, 2023

CONTENTS:

Charles Le Losq, Institut de physique du globe de Paris, University Paris Cité lelosq@ipgp.fr

Barbara Baldoni, Institut de physique du globe de Paris, University Paris Cité baldoni@ipgp.fr

Andrew P. Valentine, University of Durham andrew.valentine@durham.ac.uk

CHAPTER

ONE

THE I-MELT PROJECT

i-Melt is a physics-guided neural network model, that combines deep neural networks with physical equations to predict the structural, thermodynamic and dynamic properties of aluminosilicate melts and glasses.

The project is hosted on Github, a Streamlit web calculator is available and you can read the papers explaining the model here and here!

1.1 Installation

1.1.1 General preparation

i-Melt runs with a traditional Python stack.

If you are not familiar with Python, you can first have a look at the scipy lecture notes, a set of tutorials for the beginner.

You can install Anaconda Python to get a running Python distribution. See the documentation of Anaconda for those steps.

1.1.2 Installation of libraries for i-Melt

The necessary libraries are listed in the requirements.txt file.

Scripts for building, training models and providing useful functions are contained in the src folder.

Starting from a barebone Python 3 environment with pip installed, simply open a terminal pointing to the working folder and type:

```
$ pip install --user -r requirements.txt
```

1.2 Database

1.2.1 Database localisation

All data are given in a Database.xlsx file in the data folder.

The data used for training the currently provided mdels are in HDF5 format in the data folder.

1.2.2 Data preparation

Run the script Dataset_preparation.py to prepare the datasets, which are subsequently saved in HDF5 format in the data folder.

The Dataset_visualization.py script allows running the generation of several figures, saved in /figures/datasets/

Processed Raman spectra are also shown in /figures/datasets/raman/

1.3 Training the networks

The easiest way of training one or multiple neural networks is to use the scripts that are provided in /src.

1.3.1 Training one network

The code Training_single.py allows training only one network and playing with it. The following steps are performed.

After importing the libraries (see notebook), we load the data:

```
device = torch.device('cuda') # training on the GPU

# custom data loader, automatically sent to the GPU
ds = imelt.data_loader(device=device)
```

We select an architecture. For this example, we have selected the reference architecture from Le Losq et al. 2021:

```
nb_layers = 4
nb_neurons = 200
p_drop = 0.10 # we increased dropout here as this now works well with GELU units
```

If we want to save the model and figures in the directories *./model/candidates/* and *./figures/single/*, we can use this code to check if the folders exist, and create them if not:

```
utils.create_dir('./model/candidates/')
utils.create_dir('./figures/single/')
```

Now we need a name for our model, we can generate it with the hyperparameters actually, this will help us having automatic names in case we try different architectures:

```
name = "./model/candidates/l"+str(nb_layers)+"_n"+str(nb_neurons)+"_p"+str(p_drop)+"_test
↪"+".pth"
```

and we declare the model using *imelt.model()*:

```
neuralmodel = imelt.model(ds.x_visco_train.shape[1],
                                                    hidden_size=nb_neurons,
                                                    num_layers=nb_layers,
                                                    nb_channels_raman=ds.nb_channels_raman,
                                                    activation_function = torch.nn.GELU(),
                                                    p_drop=p_drop)
```

We select a criterion for training (the MSE criterion from PyTorch) and send it to the GPU device

```
criterion = torch.nn.MSELoss(reduction='mean')
criterion.to(device) # sending criterion on device
```

Before training, we need to initilize the bias layer using the imelt function, and we send the network parameters to the GPU:

```
neuralmodel.output_bias_init()
neuralmodel = neuralmodel.float() # this is just to make sure we are using always␣
↪float() numbers
neuralmodel.to(device)
```

Training will be done with the ADAM optimizer with a tuned learning rate of 0.0003:

```
optimizer = torch.optim.Adam(neuralmodel.parameters(), lr = 0.0003)
```

We have build a function for training in the imelt library that performs early stopping. You have to select:

- the patience (how much epoch do you wait once you notice the validation error stop improving)
- the min_delta variable, that represents the sensitivity to determine if the RMSE on the validation dataset really improved or not

The *imelt.training()* function outputs the trained model, and records of the training and validation losses during the epochs.

Training can thus be done with this code:

```
neuralmodel, record_train_loss, record_valid_loss = imelt.training(neuralmodel,ds,
                criterion,optimizer,save_switch=True,save_name=name,
                train_patience=250,min_delta=0.05,
                verbose=True)
```

1.3.2 Hyperparameter tuning

RAY TUNE + OPTUNA

In the version 2.0, we rely on Ray Tune and Optuna to search for the best models.

The script ray_opt.py allows running a Ray Tune experiment.

The script ray_select.py allows selecting the best models based on posterior analysis of the Ray Tune experiment (all metrics recorded in an Excel spreadsheet that must be provided for model selection).

Bayesian optimization

CURRENTLY NOT WORKING

The bayesian_optim.py script allows performing Bayesian Optimization for hyperparameter selection using AX plateform.

1.3.3 Training candidates

Note : this was used in v1.2 for model selection, but now we rely on the Ray Tune + Optuna run to select models.

In any case, this still works. The code Training_Candidates.py allows training 100 networks with a given architecture and selects the 10 best ones, which are saved in ./model/best/ and used for future predictions.

1.4 Result analysis

IN CONSTRUCTION...

All figures are saved in i-melt/figures

1.5 Predictions

1.5.1 Web calculator

The easiest way to try i-Melt is to use the web calculator.

1.5.2 Jupyter notebooks

More control can be achieved using directly the i-melt library.

Several example notebooks are provided in the main repository. We invite you to have a look at them directly.

If you want to have an example of use for making predictions for a given composition, have a look at the Example_Prediction_OneComposition.ipynb notebook.

The steps are simple. First, import the necessary libraries and imelt:

```
%matplotlib inline #matplotlib magic


import numpy as np # for arrays
import pandas as pd # for dataframes
import matplotlib.pyplot as plt # for plotting
import src.imelt as imelt # imelt core functions
import src.utils as utils # utility functions
```

Then, we can load the pre-trained i-melt models as one Python object:

```
imelt_model = imelt.load_pretrained_bagged()
```

Now we can define a composition of interest in a Panda dataframe. We also automatically add descriptors to the composition.

```
composition = [0.75, # SiO2
               0.125, # Al2O3
               0.125, # Na2O
               0.0, # K2O
               0.0, # MgO
               0.0] # CaO
```

(continues on next page)

(continued from previous page)

```
# we transform composition in a dataframe and add descriptors
composition = pd.DataFrame(np.array(composition).reshape(1,6), columns=["sio2","al2o3",
↪"na2o","k2o","mgo","cao"])
composition = utils.descriptors(composition.loc[:,["sio2","al2o3","na2o","k2o","mgo","cao
↪"]]).values
```

Predictions can be obtained for Tg with:

```
tg = imelt_model.predict("tg", composition)
```

If you want error bars, you need to ask for samples:

```
tg = imelt_model.predict("tg", composition, sampling=True, n_sample=20)
```

Here tg contains 20 samples from the 10 different models, so a total of 200 predictions. You can now calculate the standard deviation and mean values of tg as:

```
tg_standard_deviation = np.std(tg)
tg_mean = np.mean(tg)
```

Another way, better, may be to ask for the 95% confidence intervals and the median:

```
tg_95CI = np.percentile(tg, [2.5, 97.5])
tg_median = np.median(tg)
```

We can predict the viscosity with the Vogel-Tammann-Fulscher equation. First, we create a array containing the temperatures of interest, then we calculate the viscosity:

```
T_range = np.arange(600, 1500, 1.0) # from 600 to 1500 K with 1 K steps
viscosity = imelt_model.predict("tvf",composition*np.ones((len(T_range),39)),T_range.
↪reshape(-1,1))
```

In the above code note that the composition array has to be modified so that you have as many lines as you have temperatures to predict.

Many other predictions are possible, look at the Jupyter notebooks for more details and examples.

1.6 References

Le Losq C., Valentine A. P., Mysen B. O., Neuville D. R., 2021. Structure and properties of alkali aluminosilicate glasses and melts: insights from deep learning. Geochimica and Cosmochimica Acta, https://doi.org/10.1016/j.gca.2021.08.023

Le Losq C., Valentine A. P., Baldoni B., 2021. From Windows to Volcanoes: How PyTorch Is Helping Us Understand Glass. Medium/pytorch, https://medium.com/pytorch/from-windows-to-volcanoes-how-pytorch-is-helping-us-understand-glass-8720d480f4f2

Le Losq C., Valentine A. P., 2021. charlesll/i-melt: i-Melt v1.2.1 (v1.2.1). Zenodo. https://doi.org/10.5281/zenodo.5342178

Le Losq C., Baldoni B., Valentine A. P., 2023. charlesll/i-melt: i-Melt v2.0.0 (v2.0.0). Zenodo. https://doi.org/10.5281/zenodo.7858297

Le Losq C., Badoni B., 2023. Machine learning modeling of the atomic structure and physical properties of alkali and alkaline-earth aluminosilicate glasses and melts. arXiv:2304.12123 https://doi.org/10.48550/arXiv.2304.12123

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search