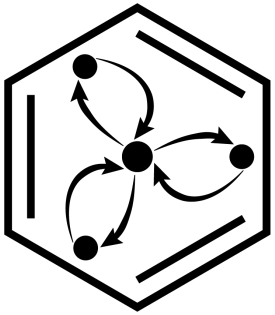


chemprop

Datasets and Machine Learning Software for Chemical Property Prediction

Kevin P. Greenman, Haoyang (Oscar) Wu, and William H. Green

16 August 2023

ACS Fall 2023 Meeting

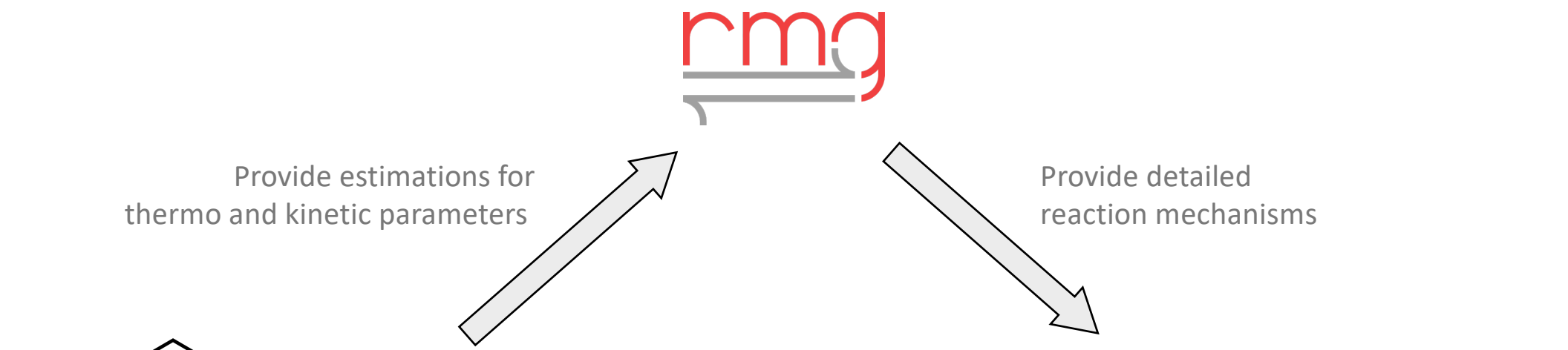
CATL Open-Source Software Workshops

Massachusetts Institute of Technology - Department of Chemical Engineering

Massachusetts Institute of Technology

Reaction Mechanism Generator

Automatic construction of kinetic mechanisms



chemprop

Chemprop

Machine learning package
for chemical and reaction property prediction

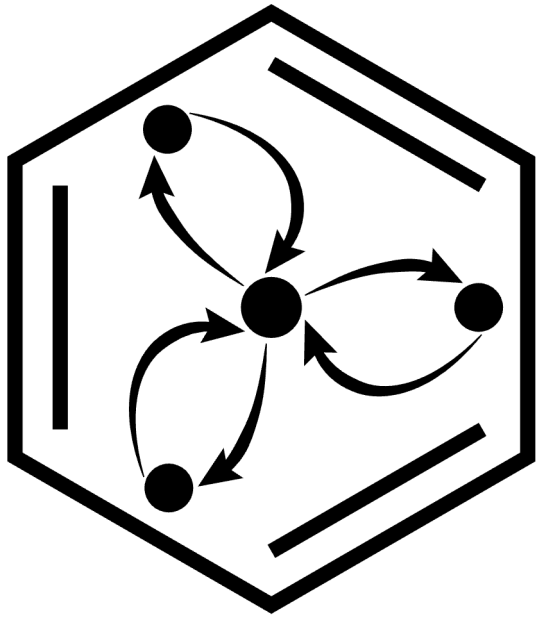
rms

Reaction Mechanism Simulator

Simulate and analyze
large chemical reaction mechanisms

What we'll cover today

- Successful use cases
- Chemprop algorithm overview
- Data input formatting
- Training
- Prediction
- Open-source datasets
- Interactive tutorial (Colab notebook)



Chemprop – Community Adoption

Chemprop on GitHub:

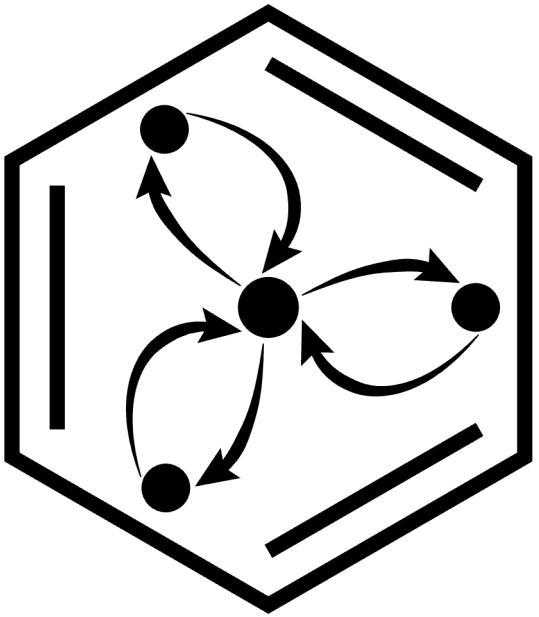
1,155 stars

#1 repo in “chemistry” topic

2,133 PyPI downloads in last month

54,533 lifetime PyPI downloads

11,500 lifetime conda downloads



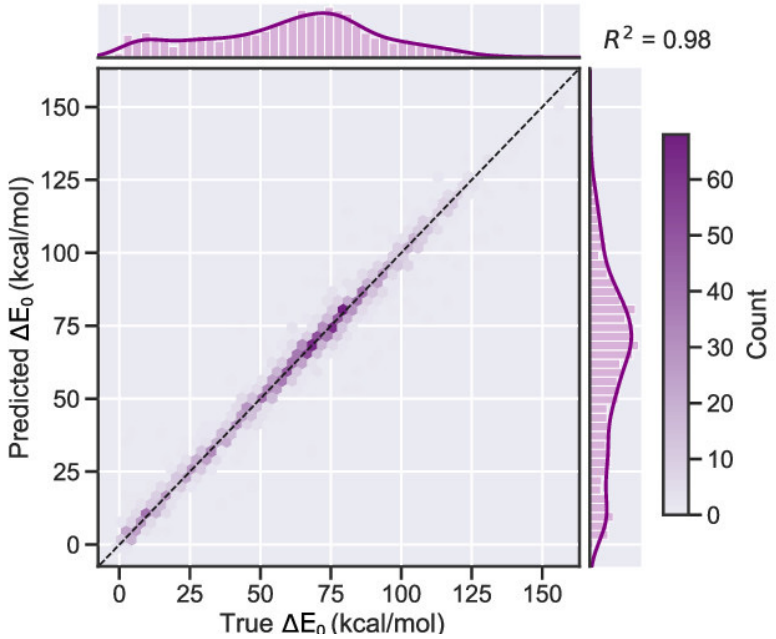
Successful Use Cases

Why should you learn to use Chemprop?

Massachusetts Institute of Technology

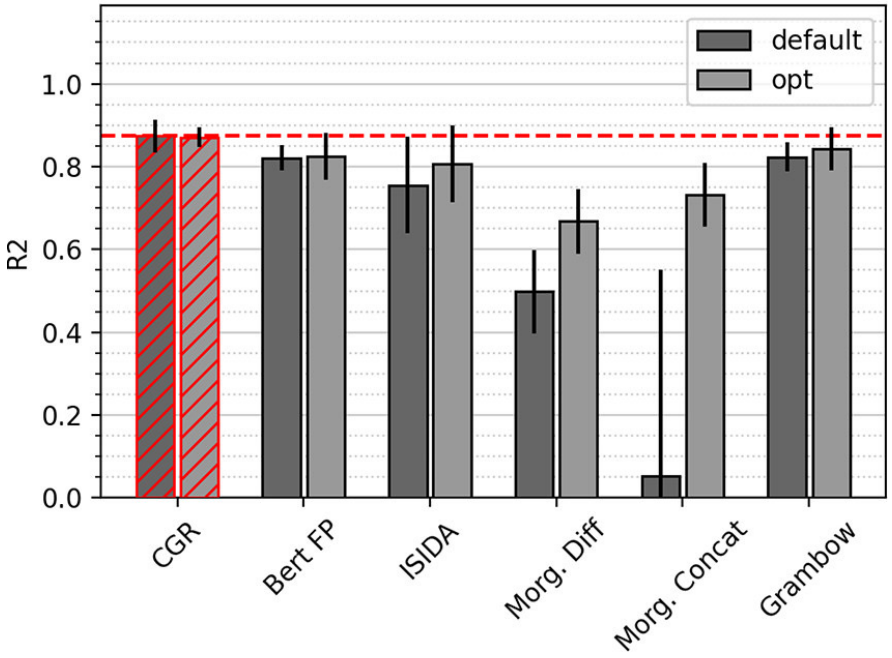
Chemprop for reaction barrier height prediction

Trained using
quantum calculation data



K. Spiekermann, L. Pattanaik, W. H. Green,
J. Phys. Chem. A. 126 (2022) 3976-3986

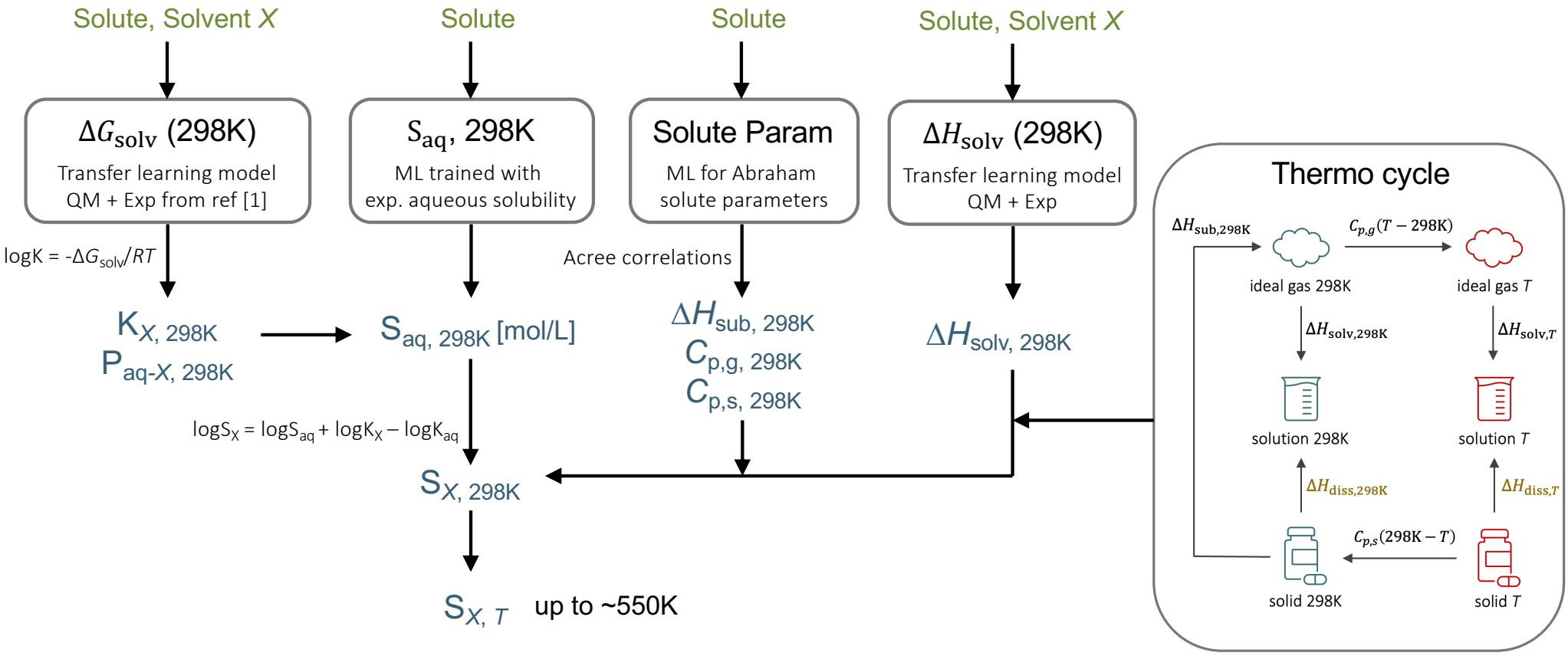
Activation energy for S_NAr reactions



E. Heid, W. H. Green,
J. Chem. Inf. Model. 62 (2022) 2101-2110

Massachusetts Institute of Technology

Solubility prediction by combining several ML models with thermodynamic equations

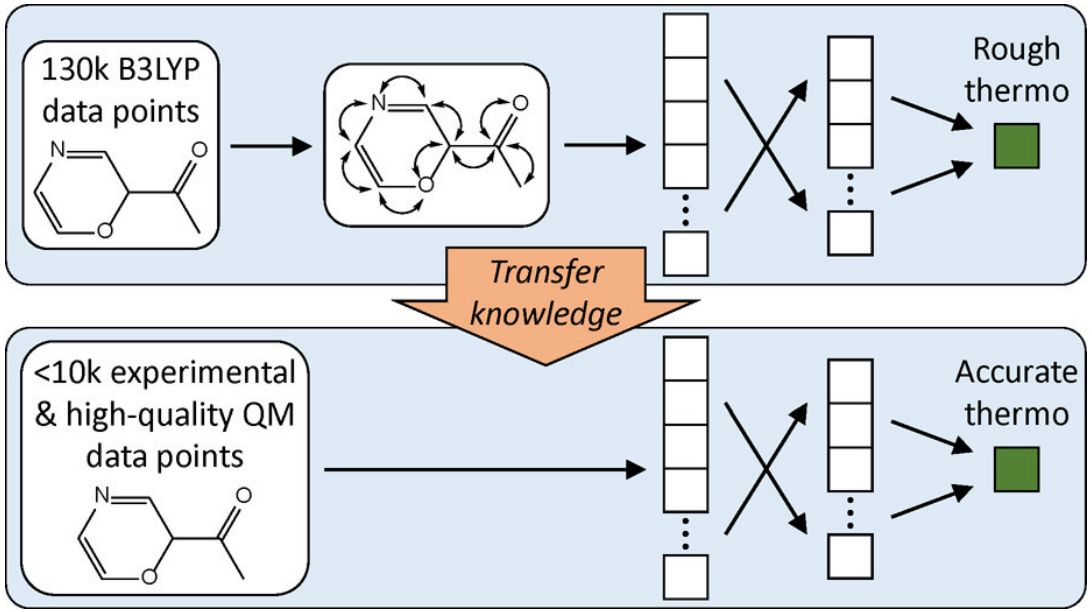


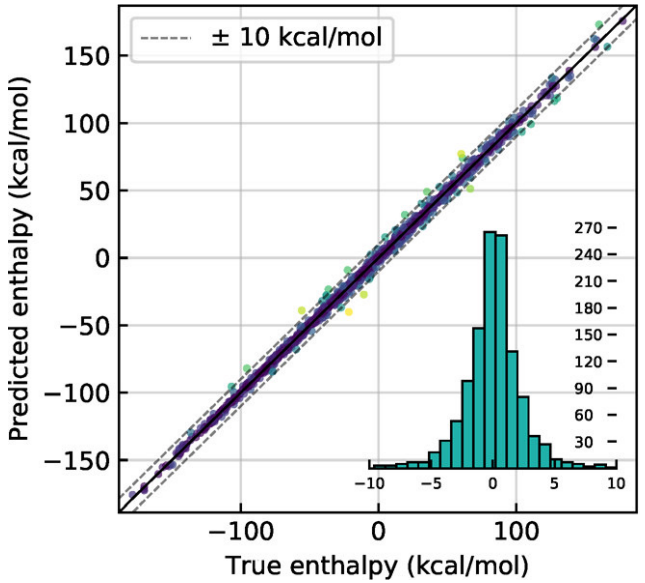
F. Vermeire, Y. Chung, W. H. Green, *J. Am. Chem. Soc.* 144 (2022) 10785-10797

Find some data from literature → Train a ML model → Publish it!

Find data, Physical laws, Thermodynamic relationships, Data cleaning, ML design choice, Generate data through ab initio calculations, Compound Representations, Feature choice → applications → Publish it!

Thermochemistry prediction using QM calculations and experimental data



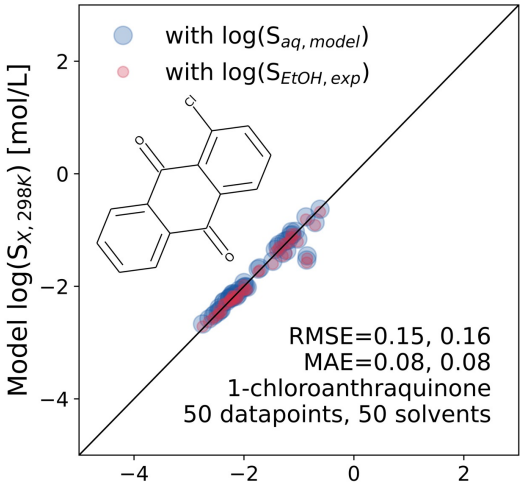


Predict enthalpy, entropy, and heat capacity of species

C. Grambow, Y-P. Li, W. H. Green, *J. Phys. Chem. A.* 123 (2019) 5826-5835

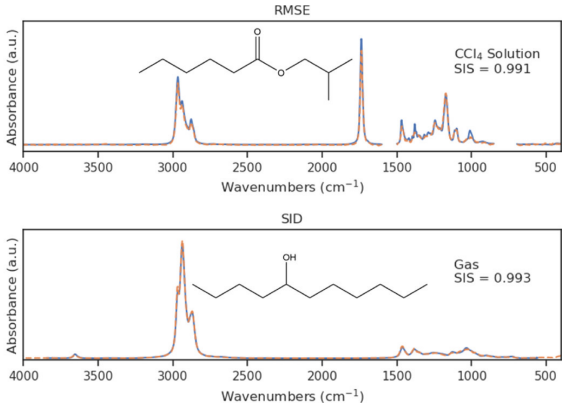
Chemprop for other chemical property predictions

Solid solubility



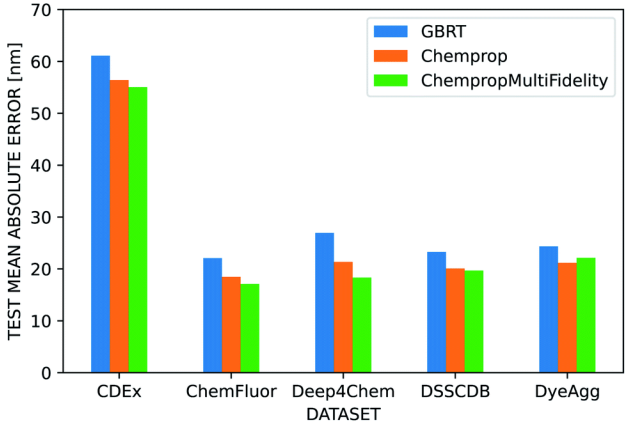
F. Vermeire, Y. Chung, W. H. Green,
J. Am. Chem. Soc. 144 (2022) 10785-10797

IR Spectra



C. McGill, M. Forsuelo, Y. Guan, W. H. Green,
J. Chem. Inf. Model. 61 (2021) 2594-2609

UV/Vis



K. P. Greenman, W. H. Green, R. Gómez-Bombarelli,
Chem. Sci. 13 (2022) 1152-1162

Algorithm Overview

How does Chemprop work?

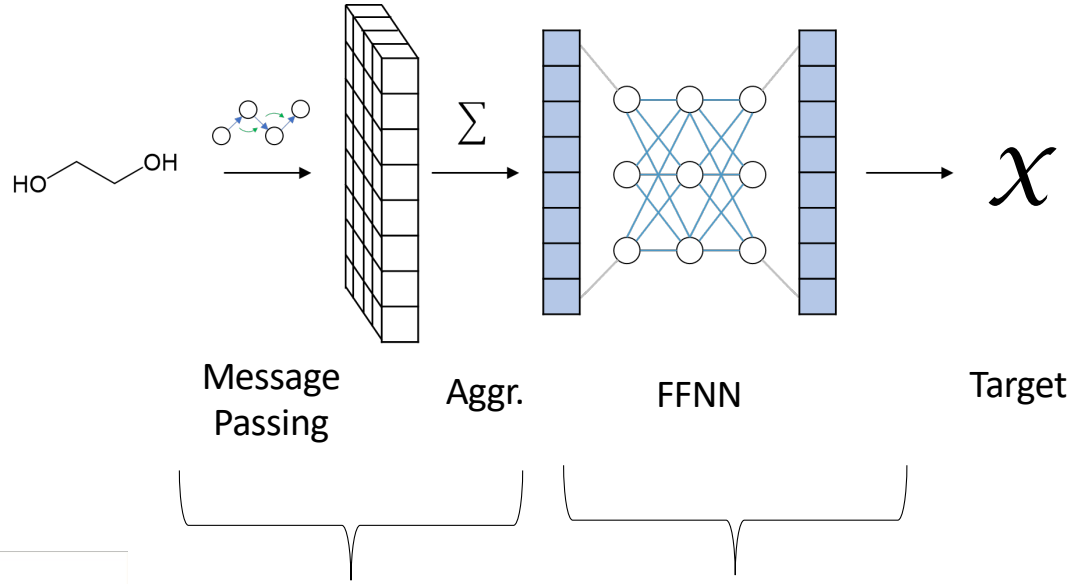
Chemprop: ML package for chemical and reaction property prediction

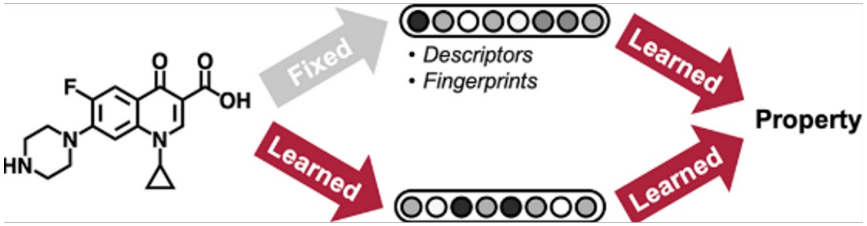
- Python-based machine learning package that is actively used, developed, and maintained by the Green Group and other groups
- Graph convolutional model with directed message passing neural network (D-MPNN)
- Easy to use, flexible architecture with various useful options
  e.g. pretraining, multiple compounds, additional features, weighted training, etc.
- Strong molecular property prediction capabilities across a range of properties:
  - Activation energy of reaction
  - Thermochemical parameters of species
  - Kinetic solvent effects
  - IR spectra
  - Many more!

K. Yang et al. *J. Chem. Inf. Model*. 2019, 59, 3370–3388.
E. Heid et al. *ChemRxiv*. 2023. DOI: 10.26434/chemrxiv-2023-3zcfl

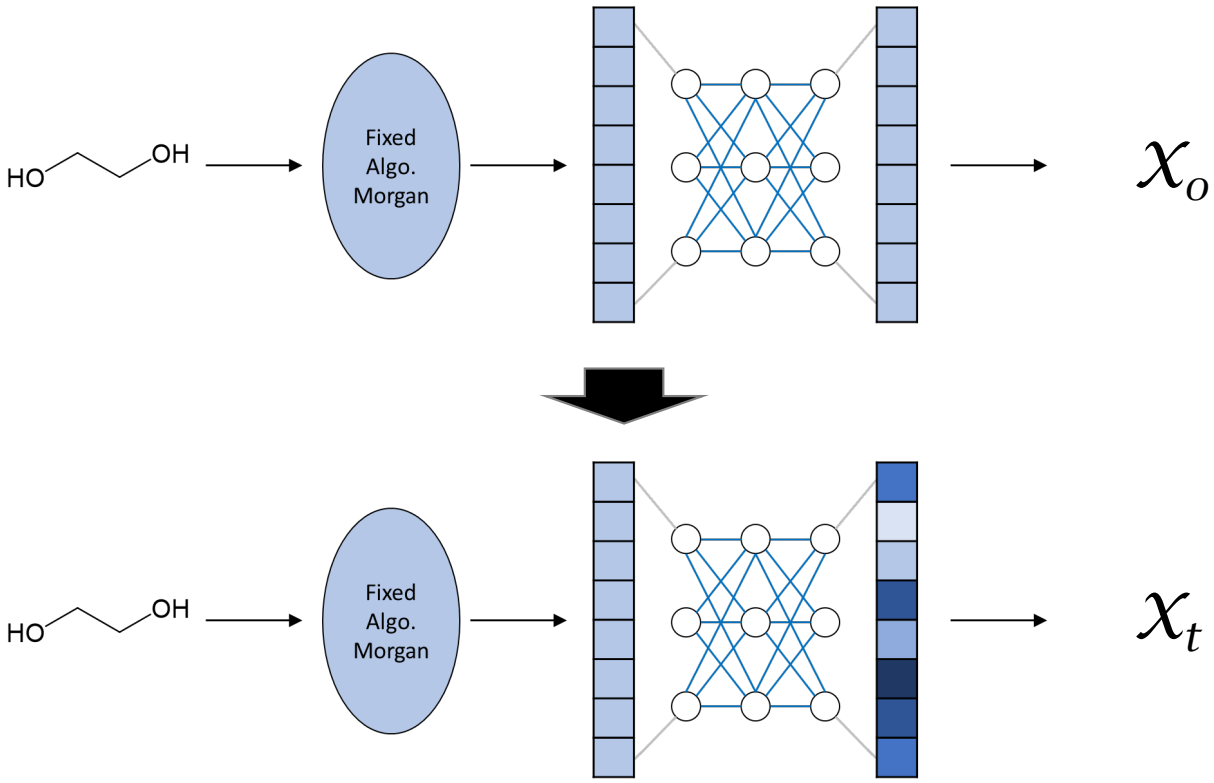
Chemprop Structure & Difference from Traditional QSAR

- Two networks.
  - Directed Message Passing Neural Network (D-MPNN)
  - Feed Forward Neural Network (FFNN)





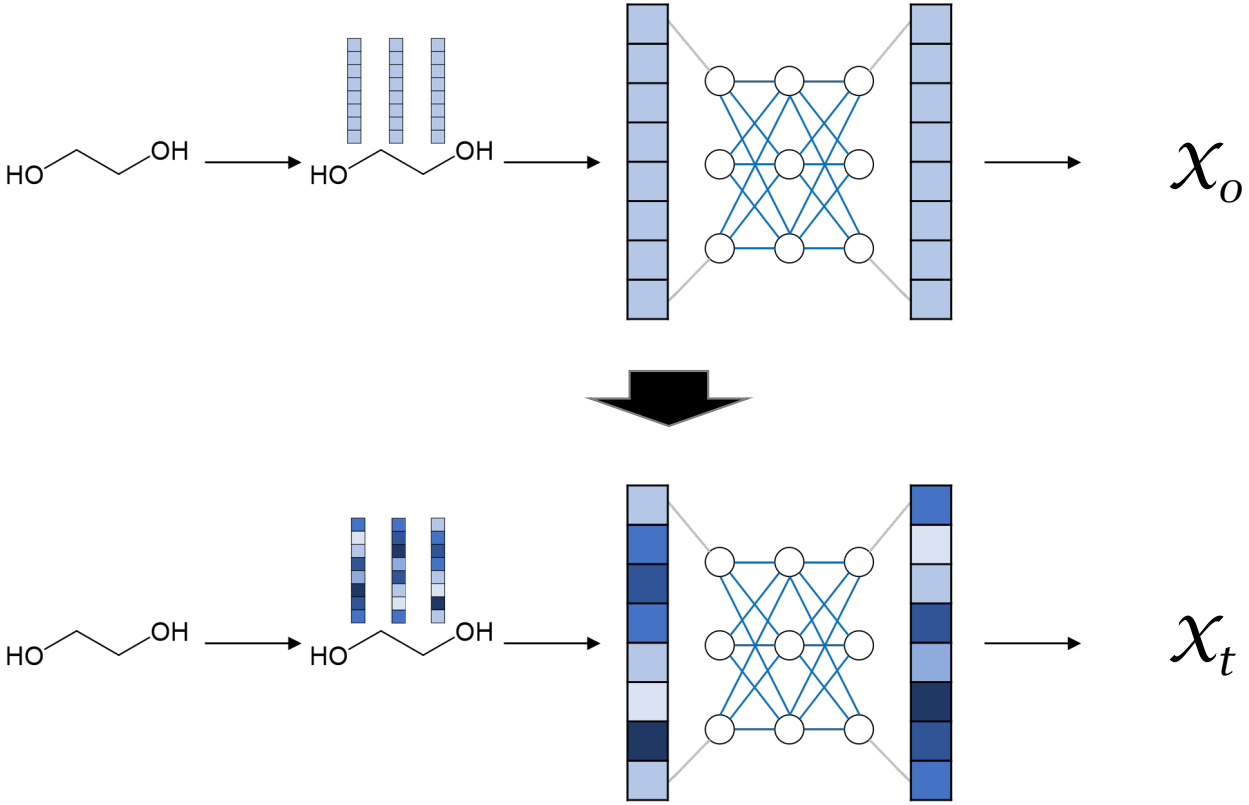
Traditional Approach: Fixed Encodings



Fixed algorithms

- General purpose molecule vectors based on molecular structure.
- Stay the same through training

MPNN – Trainable Encodings



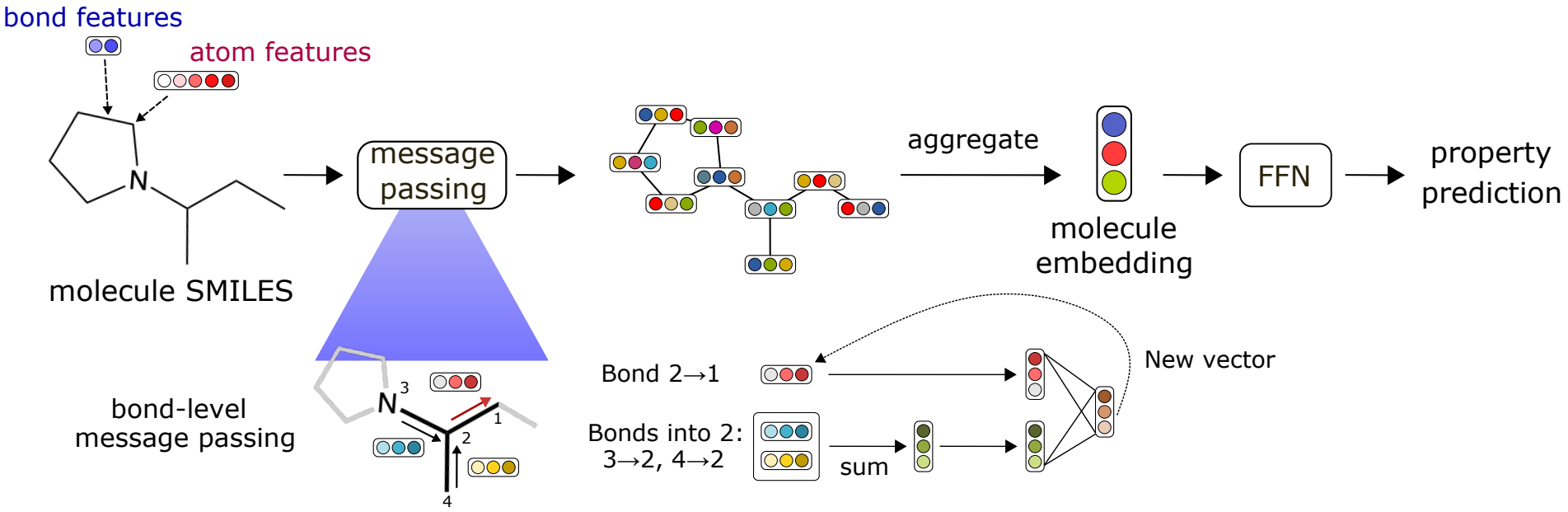
Message Passing

- Assigns a feature vector to each bond and atom.
- Updates feature vectors through differentiable operations with neighbors.
- Encoding optimized to your task.
- Learn the important features and interactions as you go.

Chemprop architecture

D-MPNN: directed message passing neural network

FFN: feed-forward neural network



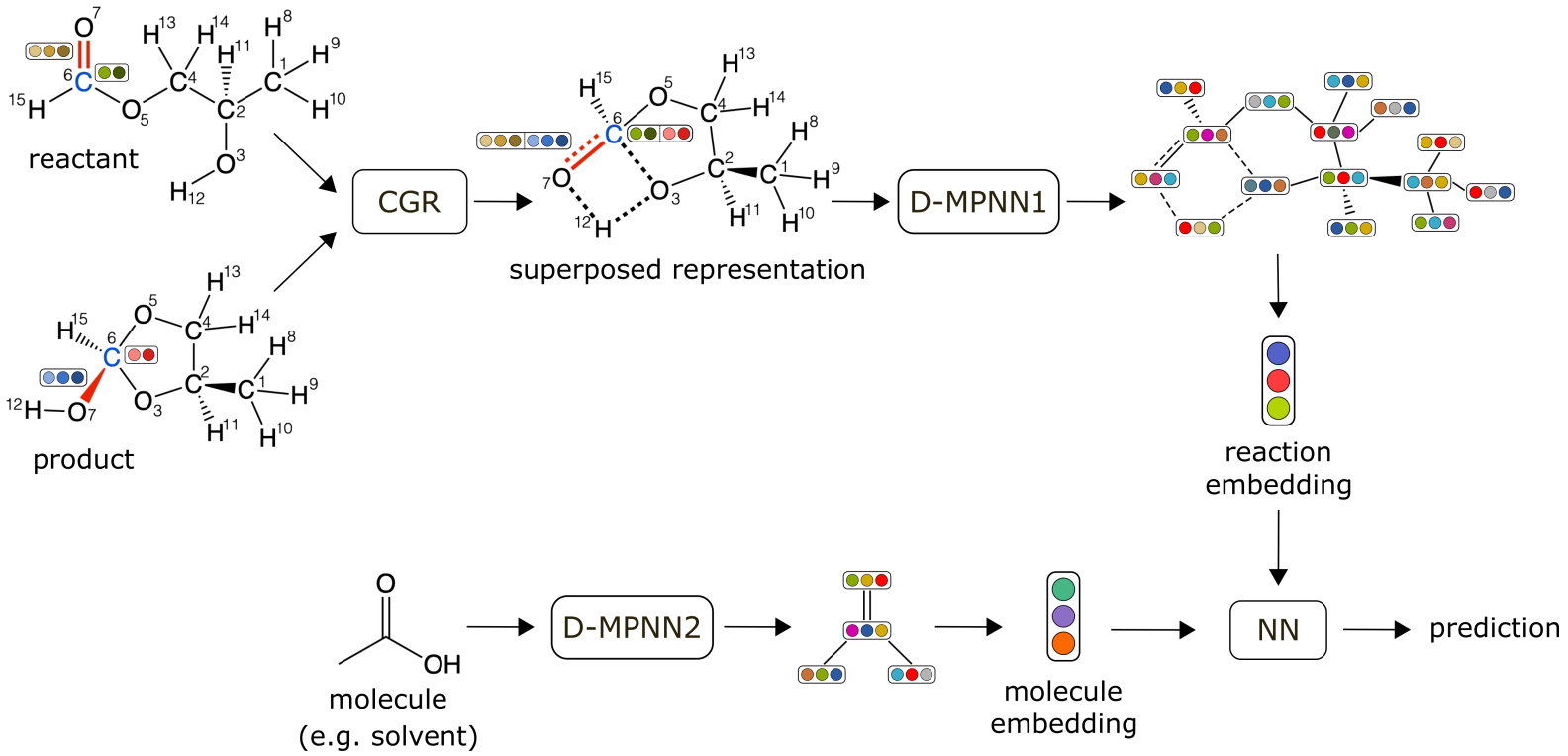
Atom features: atomic number, molar mass, aromaticity, formal charge, …

Bond features: bond type, conjugation, stereo, …

K. Yang et al. *J. Chem. Inf. Model*. 2019, 59, 3370–3388.
E. Heid et al. *ChemRxiv*. 2023. DOI: 10.26434/chemrxiv-2023-3zcfl

Chemprop for reaction property prediction

CGR: Condensed Graph of Reaction



E. Heid, W. H. Green, J. Chem. Inf. Model. 2021, 62, 2101–2110

Chemprop is a software *and* a package for molecular MPNNs

Richly featured with support for

- variety of prediction tasks
  regression, classification, multiclass, spectral
- diverse input types
  molecules, reactions, or combinations
- customizable model architectures
  bond/atom-based messages, # message-passing iterations, etc.
- robust training
  variably weighted data, missing values

```
$ chemprop_train \
    --data_path data/tox21.csv \
    --dataset_type classification \
    --save_dir tox21_checkpoints
```

```
$ chemprop_predict \
    --test_path data/tox21.csv \
    --checkpoint_dir tox21_checkpoints
    --preds_path tox21_preds.csv
```

Easy-to-use via simple command line interface

Data Input Formatting

CSV data format

- Data for the model should be stored in a csv file format.
- Include a header row.
- Molecules must be specified with a SMILES (or reaction SMILES).
- Unknown target values can be left blank.

	A	B	C	D	E	F
1	smiles_1	smiles_2	target_1	target_2	target_3	other_column
2	O=C(O)c1ccco1	BrC(Br)c1cccc(C(Br)Br)c1	8	8	3	0
3	O=C(O)c1ccco1	CCSC#N	4	8	0	7
4	c1ccc(C2=NOC(c3ccccc3)C2)cc1	CCCCCCCCCCCCCCCCCCCC	6	7	0	5
5	c1ccc(C2=NOC(c3ccccc3)C2)cc1	CC(C)CCC#N	8	7	4	7
6	Clc1ccc(OCc2ccccc2)cc1	CC(C)CCC#N	6	4	6	4
7	Clc1ccc(OCc2ccccc2)cc1	C=CCOC(=O)CCC(=O)OCC=C	6	2	4	6
8	Cc1ccc(OCc2ccccc2)cc1	CCc1ccccn1	1	4	9	1
9	Cc1ccc(OCc2ccccc2)cc1	CCCCC/C=C\CCCCC(=O)OCCCC	6	4	5	10
10	CC(=O)Oc1ccc(C=O)cc1Br	CCOC(=O)C(=O)C(C)C(=O)OCC	0	3	6	4
11	CC(=O)Oc1ccc(C=O)cc1Br	O=S(=O)(Cl)c1ccccc1	4	1	2	7
12	CN=C(O)Oc1cccc(C(=O)O)c1	C=CCc1ccc(O)c(OC)c1	6	9	10	6
13	CN=C(O)Oc1cccc(C(=O)O)c1	ClCC(Cl)CCl	0	0	10	10
14	CN=C(O)Oc1cccc(C(=O)OC)c1	C=CCc1ccc(OC(C)=O)c(OC)c1	8	9	5	6
15	CN=C(O)Oc1cccc(C(=O)OC)c1	CCc1ccccc1[N+](=O)[O-]	0	7	5	1
16	COc1ccc(OCc2ccccc2)cc1	CCc1ccccc1[N+](=O)[O-]	0	5	8	10

Two optional arguments for specifying columns in your data

--smiles_columns smiles_1 smiles_2
--target_columns target_1 target_3

Default behavior is the first n columns are smiles for however many molecules you're using, and all others are targets.

Dataset Types and Targets

Regression

- Targets are float values. With bounded loss functions or metrics, the values may also be simple inequalities (e.g., >7.5 or <5.0).

Classification

- Targets are binary (i.e., 0s and 1s) indicators of the classification.

Multiclass

- Targets are integers (starting with zero) indicating which class the datapoint belongs to, out of a total number of exclusive classes indicated with --number_of_classes <int>.

Spectra

- Targets are positive float values with each target representing the signal at a specific spectrum position.

Additional Features

- Molecular features for each datapoint can be provided in a separate csv data file.
- Features file must have a header row and contain the same number of values as the smiles data file. But the features file does not contain SMILES columns.
- Multiple features files can be provided.
- Options are also available for atom/bond specific features.

Data File

smiles_1	smiles_2	target_1	target_2
O=C(O)c1ccco1	BrC(Br)c1cccc(C(Br)Br)c1	8	8
O=C(O)c1ccco1	CCSC#N	4	8
c1ccc(C2=NOC(c3ccccc3)C2)cc1	CCCCCCCCCCCCCCCCCCCC	6	7
c1ccc(C2=NOC(c3ccccc3)C2)cc1	CC(C)CCC#N	8	7
Clc1ccc(OCc2ccccc2)cc1	CC(C)CCC#N	6	4
Clc1ccc(OCc2ccccc2)cc1	C=CCOC(=O)CCC(=O)OCC=C	6	2
Cc1ccc(OCc2ccccc2)cc1	CCc1ccccn1	1	4
Cc1ccc(OCc2ccccc2)cc1	CCCCC/C=C\CCCCC(=O)OCCCC	6	4
CC(=O)Oc1ccc(C=O)cc1Br	CCOC(=O)C(=O)C(C)C(=O)OCC	0	3
CC(=O)Oc1ccc(C=O)cc1Br	O=S(=O)(Cl)c1ccccc1	4	1
CN=C(O)Oc1cccc(C(=O)O)c1	C=CCc1ccc(O)c(OC)c1	6	9
CN=C(O)Oc1cccc(C(=O)O)c1	ClCC(Cl)CCl	0	0
CN=C(O)Oc1cccc(C(=O)OC)c1	C=CCc1ccc(OC(C)=O)c(OC)c1	8	9
CN=C(O)Oc1cccc(C(=O)OC)c1	CCc1ccccc1[N+](=O)[O-]	0	7
COc1ccc(OCc2ccccc2)cc1	CCc1ccccc1[N+](=O)[O-]	0	5

Features File

feature_1
0
7
5
7
4
6
1
10
4
7
6
10
6
1
10

Training

Input Style and Required Args

- Training
  - Input Style
- Output File Tree
- Major Hyperparameters
- Data Splitting
- Ensemble & Cross-Validation
- Loss Functions & Metrics
- Learning Rate
- Additional Parameters

```
python train.py \
--dataset_type regression \
--data_path input/data.csv \
--save_dir output/results/dir
```

- Submission from command line can be done using `chemprop_train` from anywhere or `python train.py` in the chemprop directory.
- Command should be followed by all training arguments.
- Required args:
  - dataset_type – One of the four supported types
  - data_path – Path to the data csv file
  - save_dir – Directory to store the results

The Output File Tree

Level 1 – save_dir

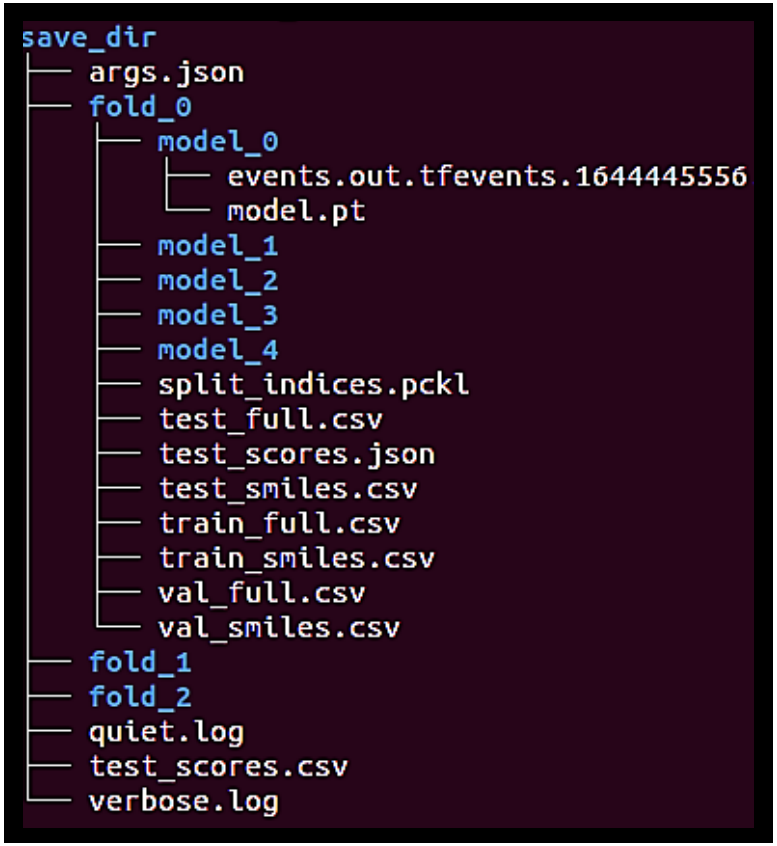
- Training arguments saved as json
- Quiet and verbose log files
- Overall test set scores

Level 2 – fold_x, cross-validation fold

- Test set scores for this fold
- Data splits if using --save_smiles_splits

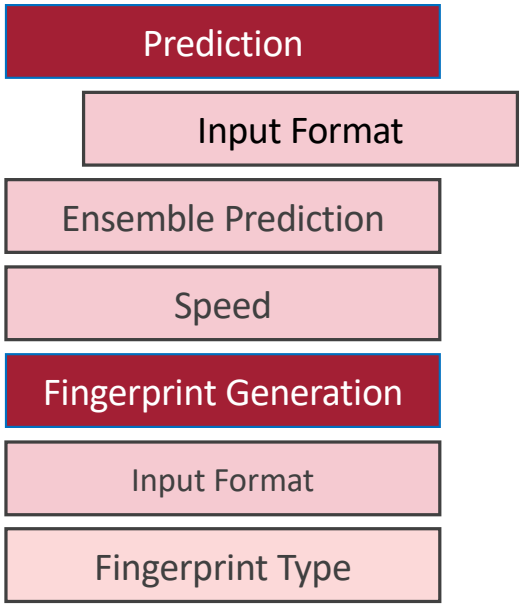
Level 3 – model_y, ensemble model

- Individual model file
- Events file for tensorboard visualization



Prediction

Input Format and Arguments




```
python predict.py \
--test_path smiles/to/predict.csv \
--preds_path path/to/save.csv \
--checkpoint_dir path/to/models/dir
```

- Can be run from command line with `chemprop_predict` or `python predict.py`.
- Most necessary arguments are imported from the loaded checkpoint files.
- Required args
  - test_path – csv file with the SMILES strings to be predicted.
  - preds_path – location to write a csv file with predictions.
  - checkpoint_dir – the directory containing the model pt files

What we haven't covered today

- Hyperparameters (choices and optimization)
- Data splitting
- Cross-validation and ensembling
- Loss functions
- Metrics
- Multi-molecule inputs
- Reproducibility
- Extracting d-MPNN fingerprints
- Uncertainty (+ calibration and evaluation)
- Pretraining / transfer learning
- Atom- and bond-level targets
- Data and task weighting
- Spectra-specific features
- Interpreting
- Random Forest and SVM
- Script utilities
- Tensorboard visualization

Datasets

Open-source reaction & molecular property datasets

RMG database

- **Thermo**: 45 libraries with **4564** entries, GAV for 1580 groups, and AEC/BAC for various LOT
- **Kinetics**: 92 libraries with **>21,000** reaction rates + rate rules
- **Solvation**: Abraham parameters for **195** pure solvents and **152** solutes + estimated values from ML.

Optical property

- TD-DFT calculations of **~28,000** dye-like molecules
- **UV/vis spectra** at ωB97X-D3/def2-SVPD level of theory

Barrier heights and TS

- Barrier with **high accuracy**: CCSD(T)-F12a/cc-pVDZ-F12//ωB97X-D3/def2-TZVP for **22,000** species and **12,000** gas phase reactions

Geometry

Energy

Massachusetts Institute of Technology

Open-source solvation property datasets

pKa dataset

- Includes **20,000+** experimental pKa data covering nearly **9,000** molecules

Critical property

- Critical properties and acentric factors of fluids for **5539** compounds

Solubility

- SolProp: **>5000** experimental solid solubility values + model
- ML models to predict Abraham solvation parameters. Dataset include **8366** solute parameters, **20,253** solvation free energies, and **6322** solvation enthalpies

SolProp

Abraham ML

Conclusion

A new era for Chemprop (v2.0.0)…coming late 2023!

Chemprop2 is modular and extensible:

- utilize message passing blocks in novel model architectures
- custom loss functions

Chemprop2 uses PyTorch Lightning

- eliminates “boilerplate” code and enables sklearn-type `fit()` and `predict()` calls
- enables multi-GPU training

Chemprop2 has reduced the footprint of the codebase by over 50%

Chemprop2 has improved featurization throughput by 35-50%

[1] formatting with `Black` inflates line counts

Resources

- Code: https://github.com/chemprop/chemprop
  - Open GitHub *Issues* & *Pull Requests* with bug reports, enhancement requests, and questions
  - README.md is most up-to-date source of documentation
  - Readthedocs: https://chemprop.readthedocs.io/en/latest/ (currently out-of-date)
- Papers:
  - "Analyzing Learned Molecular Representations for Property Prediction", *JCIM* (2019).
  - "Chemprop: Machine Learning Package for Chemical Property Prediction", *ChemRxiv* (2023).
- Coming soon…
  - Version 2.0: https://github.com/chemprop/chemprop/tree/v2/dev
  - More tutorials and example notebooks

GitHub Repo

2019 Paper

2023 Paper

Acknowledgements

Faculty

Regina Barzilay
Connor Coley
William Green
Tim Jamison
Tommi Jaakkola
Klavs Jensen

Developers

David Graff
Kevin Greenman
Shih-Cheng Li
Oscar Wu
Esther Heid
Charles McGill
Max Liu
Kyle Swanson
Kevin Yang

Contributors

Camille Bilodeau
Yunsie Chung
Adam Fisch
Yanfei Guan
Shomik Verma
Florence Vermiere
Daniel Xu
… and many more!

chemprop

MLPDS

AMGEN · BASF We create chemistry · BAYER · Bristol Myers Squibb · gsk · janssen · LEO · Lilly · MERCK · NOVARTIS · Pfizer · sunovion · syngenta · 药明康德 WuXi AppTec

Interactive Tutorial

Chemprop Tutorial – Colab Notebooks

Demo:

http://bit.ly/chemprop-acs23

Exercises:

https://bit.ly/chemprop-acs23-ex

[1] formatting with `Black` inflates line counts

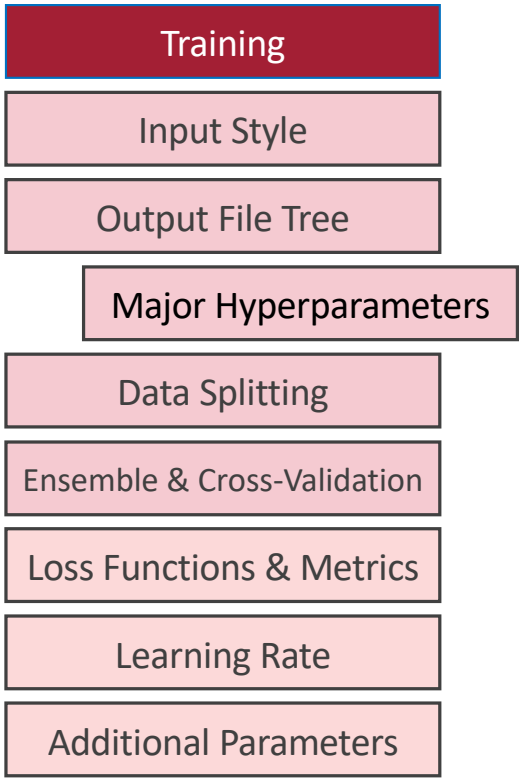
Supplementary Slides

Green Group Chemprop Tutorial – January 2023

Massachusetts Institute of Technology

Training

Major Hyperparameters

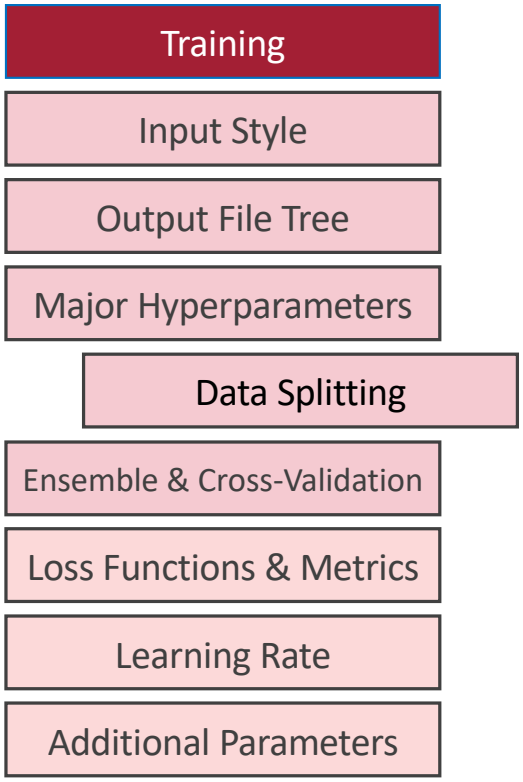


These are the hyperparameters explored in hyperparameter optimization.

- depth – number of message passing steps in MPNN.
- hidden_size – vector size used in MPNN.
- ffn_num_layers – number of layers in the FFNN.
- ffn_hidden_size – vector size used in FFNN.
- dropout – ratio used for dropout regularization. Models are often sensitive to this. Be sure to try 0 as a possible value.

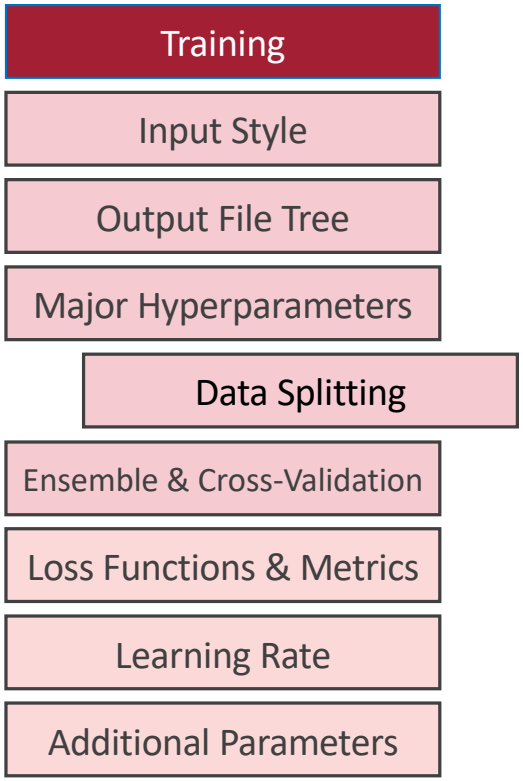
*check documentation for convenient keywords

Data Splitting



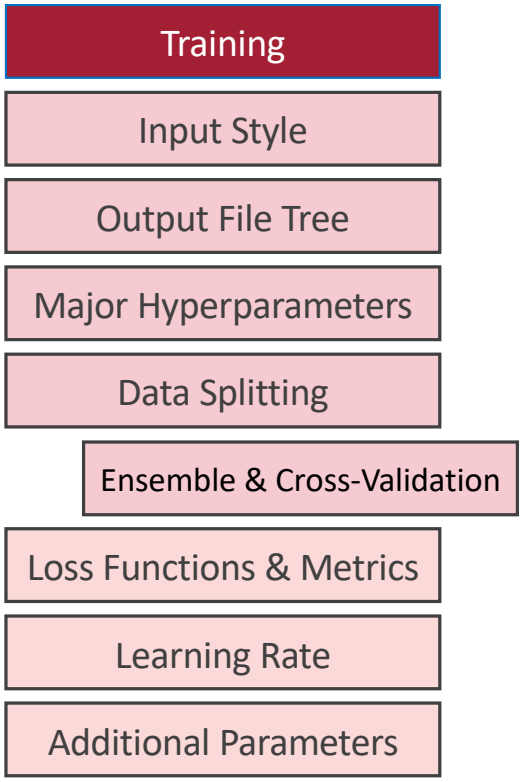
- Data provided is split into three datasets
  - **Training** – used to learn model parameters by gradient descent from the loss function.
  - **Validation** – used to determine the optimum epoch to save the model from the metric.
  - **Test** – used to evaluate the model effectiveness from the metric.
- Size ratio of the datasets can be adjusted with the argument `--split_sizes <tr> <val> <tst>`
- The splits used will be saved to csv files if you use the argument flag `--save_smiles_splits`.

Data Splitting



- Data can be split a few different ways
  - **random** – randomly assigned data.
  - **scaffold_balanced** – random but constrained that molecules with the same Murcko scaffold must be in the same split.
  - **random_with_repeated_smiles** – constrained that molecules with the same SMILES must be in the same split.
  - **cv –** a split to use with cross-validation. Ensures that each datapoint is in the test set exactly once.
  - **Provided separately** – val and test splits can be provided separately from the main data file using the arguments `--separate_test_path` and `--separate_val_path`.

Cross-Validation and Ensembling



- Cross-validation and ensembling are two multi-model options available for training.
- Cross-validation differentiates models by using different data splits. Use with `--num_folds <int>`.
- Ensemble models are differentiated by using the same data splits but different network weight initiations. Use with `--ensemble_size <int>`.
- Can be used together. For example, a 5x5 configuration would involve 25 submodels total.

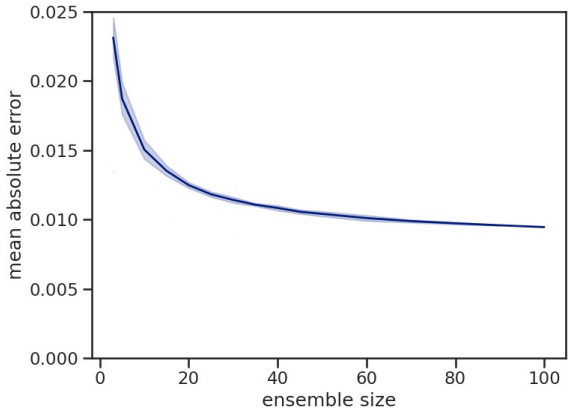
Cross-Validation and Ensembling

- Training
- Input Style
- Output File Tree
- Major Hyperparameters
- Data Splitting
- Ensemble & Cross-Validation
- Loss Functions & Metrics
- Learning Rate
- Additional Parameters

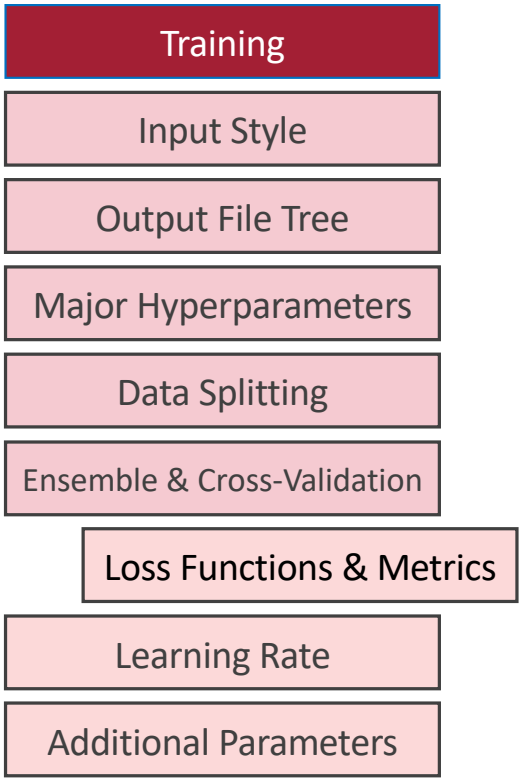
- Using multi-model approaches reduces the overall error in predictions down to an asymptote.
- The benefit comes from averaging together the predictions of the individual models before evaluating the loss/metric.
- Reduces noise in hyperparameter search.
- Both cv and ensemble have this effect on predictions for new data. But because cv doesn't have a shared test set from training to average the predictions over, it doesn't report the full benefit.

This example is an exceptional case with 50% reduction. But it is quite typical to see error reduction on the order of 5-10%, with most of the benefit from the first 5 models.

Recommendation: Use ensembles of at least 5 in your final model. If the difference from 1 model to 5 is significant, try more. Also use cross-validation if your dataset is small.

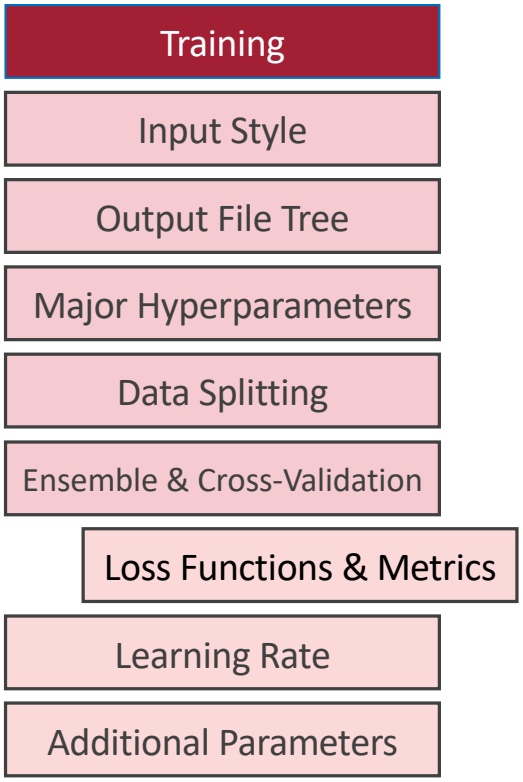


Loss Functions



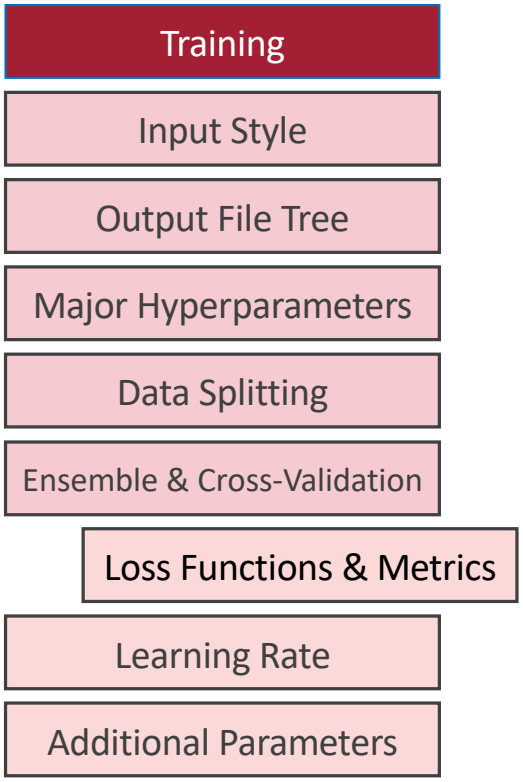
- **Loss functions** depend upon the selected dataset type. Loss functions other than defaults can be selected with the argument `--loss_function <function>`.
  - **Regression**: mse (default), bounded_mse.
  - **Classification**: binary_cross_entropy (default), mcc (a soft version of Matthews Correlation Coefficient)
  - **Multiclass**: cross_entropy (default), mcc (a soft version of Matthews Correlation Coefficient)
  - **Spectra**: sid (default, spectral information divergence), wasserstein (First-order Wasserstein distance a.k.a. earthmover's distance.)
- The **bounded** functions are functions that accept inequality targets (i.e. >5.0). Predictions which exceed inequality targets (i.e. predictions larger than 5.0) are not penalized.

Metrics



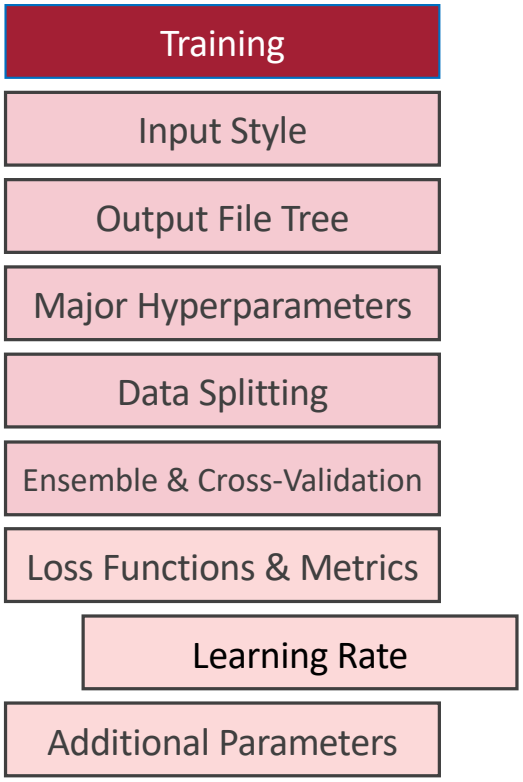
- **Metrics** are used to evaluate the success of the model against the test set as the final model score and to determine the optimal epoch to save the model at based on the validation set.
- The **primary metric** used for both purposes is selected with the argument `--metric <metric>`.
- **Additional metrics** for test set score only can be added with `--extra_metrics <metric1> <metric2> …`.

Metrics



- Supported **metrics** depend upon the dataset type.
  - **Regression.** rmse (default), mae, mse, r2, bounded_rmse, bounded_mae, bounded_mse (default if bounded_mse is loss function).
  - **Classification.** auc (default), prc-auc, accuracy, binary_cross_entropy, f1, mcc.
  - **Multiclass.** cross_entropy (default), accuracy, f1, mcc.
  - **Spectra.** sid (default), wasserstein.
- The **bounded** functions are functions that accept inequality targets (i.e. >5.0). Predictions which exceed inequality targets (i.e. predictions larger than 5.0) are not penalized.

Learning Rate



- Chemprop utilizes a **Noam Scheduler** for learning rates.
  - **Warmup**: The learning rate **increases linearly** from init_lr to max_lr during warmup steps.
  - **Remaining Steps**: The learning rate **decreases exponentially** thereafter from max_lr to final_lr.
- Users can customize the Noam Scheduler by adjusting the following arguments:
  - **Learning Rates**
    - `--init_lr <float> --max_lr <float> --final_lr <float>`
  - **Total Number of Epochs**: `--epochs <int>`
  - **Warmup Epochs**: `--warmup_epochs <int>`

Recommendation: Test longer epoch models than the default of 30, which may be sufficient for small datasets and often not for large ones.

Aggregation

- Training
- Input Style
- Output File Tree
- Major Hyperparameters
- Data Splitting
- Ensemble & Cross-Validation
- Loss Functions & Metrics
- Learning Rate
- Additional Parameters

- Atom-level representations from message passing network are averaged over all atoms of a molecule to generate a **molecule-level representation** by default.
- Alternatively, atomic vectors can be **aggregated** by:
  - summation: `--aggregation sum`
  - summation followed by division with constant N: `--aggregation norm --aggregation_norm <N>`
- Mean aggregation is most appropriate when properties are intensive, not scaling with the number of atoms in a molecule.
- Norm aggregation is most appropriate when properties are extensive, scaling directly with the number of atoms.
- Most properties are a combination of the two. Current recommendation is to use norm aggregation to start and test during development a comparison between norm and mean for your model.

Multiple Molecules

- Training
- Input Style
- Output File Tree
- Major Hyperparameters
- Data Splitting
- Ensemble & Cross-Validation
- Loss Functions & Metrics
- Learning Rate
- Additional Parameters

- **Multiple molecules** can be considered in a **single datapoint**.

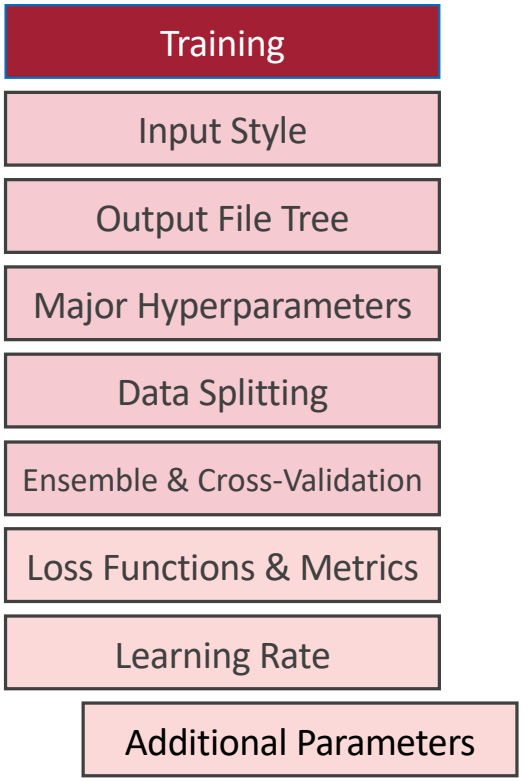
Table 1. Example of dataset with multiple molecules per datapoint; Example illustrates datapoint mapping Diels-Alder reactants to an enthalpy of reaction and activation energy in kcal/mol [Bach].

smiles_1	smiles_2	dH298	Ea298
C=CC=C	C=C	-49.1	25.4
⋮			

- Users should specify the number of molecules per datapoint through the keyword `--number_of_molecules <N>`.
- Users can specify whether a shared MPN is used to map multiple molecules for a single datapoint. The flag is `--mpn_shared`. Otherwise, Chemprop defaults to using a separate MPN for each of the number of molecules specified.

Bach, R. D.; McDouall, J. J. W.; Schlegel, H. Bernhard. Electronic Factors Influencing the Activation Barrier of the Diels-Alder Reaction. An Ab Initio Study. *J. Org. Chem.* **1989**, 54, 2931-2935

Reproducibility



- The **random seed for splitting** training, validation, and test sets can be specified with the argument `--seed <int>`.
- The **random seed for PyTorch** can be specified with the argument `--pytorch_seed <int>`.

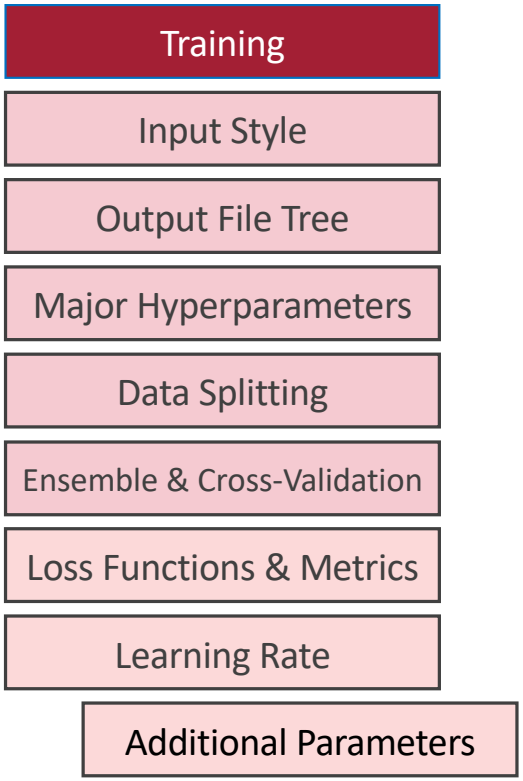
Recommendation: Manipulation of these seeds can be used to manually construct cross-validation and ensemble submodels incrementally or in parallel. If done this way, the user should test individual predictions of the submodels to ensure that no two models are accidentally identical.

GPUs

- Training
- Input Style
- Output File Tree
- Major Hyperparameters
- Data Splitting
- Ensemble & Cross-Validation
- Loss Functions & Metrics
- Learning Rate
- Additional Parameters

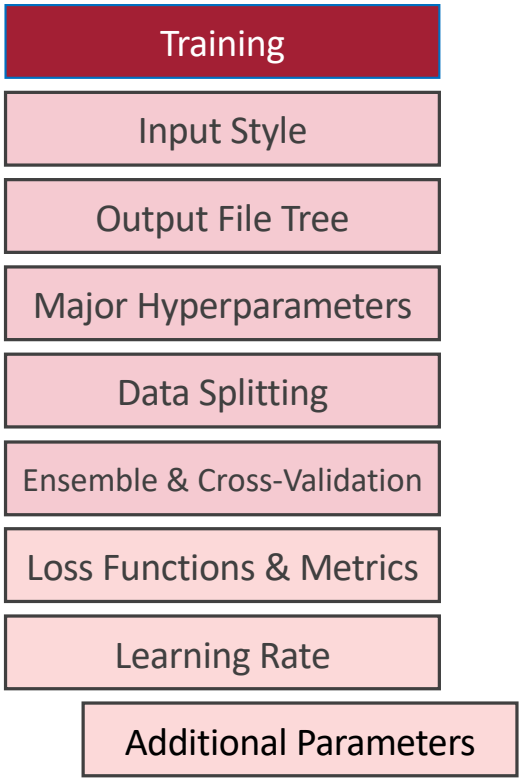
- **GPUs** can be used to accelerate training Chemprop models, particularly for larger datasets and models. The Chemprop software makes use of PyTorch's CUDA compute capabilities.
- The following dependencies are required for GPU usage:
  - cuda >= 8.0
  - cuDNN
  - possible separate install of cudatoolkit and torchvision dependencies. https://pytorch.org/get-started/locally/
- Chemprop will use available cuda-enabled nvidia GPUs by default.
- Users should specify which GPU to use by non-negative index with keyword `--gpu <int>`.
- Users can disable CUDA compute compatibilities with flag `--no_cuda`.

Additional Features



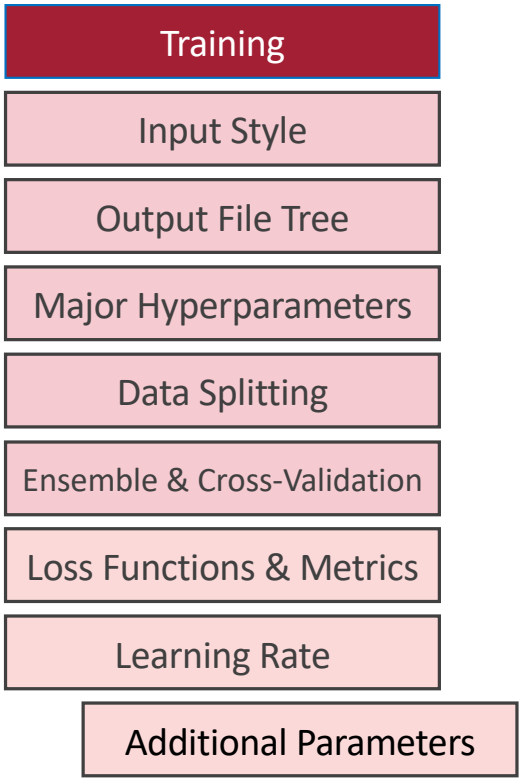
- Additional machine-learning features can be added at the atom-, bond-, or molecule-level.
- **Molecule-Level Custom Features**
  - **Generating features**: Features can be generated in code by writing a custom features generator function in `chemprop/features/features_generators.py`.
  - **Load features**: Features saved in numpy or csv files can be loaded with the argument `--features_path /path/to/features`. By default, loaded features will be normalized.
- Disable feature scaling with flag `--no_features_scaling`.

Additional Features



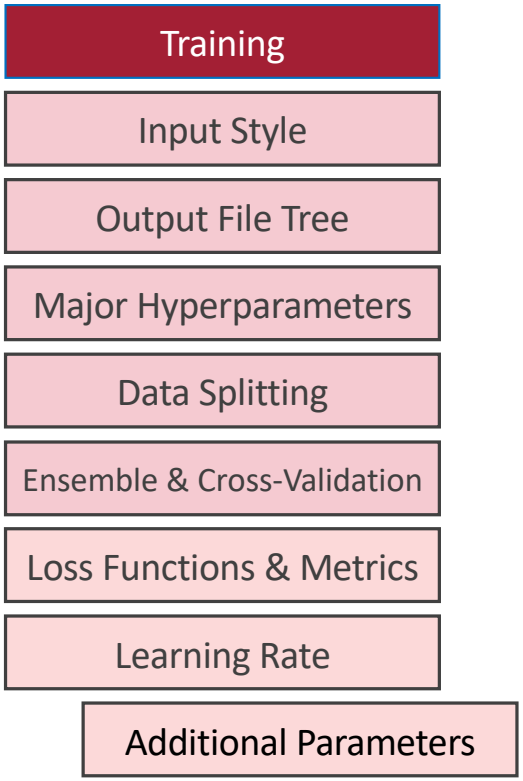
- Additional machine-learning features can be added at the atom-, bond-, or molecule-level.
- **Molecule-Level RDKit 2D Features**
  - The full list of features for argument `--features_generator` is as follows:
    - morgan: binary Morgan fingerprints, radius 2 and 2048 bits
    - morgan_count: count-based Morgan, radius 2 and 2048 bits
    - rdkit_2d: unnormalized version of 200 assorted rdkit descriptors
    - rdkit_2d_normalized: CDF-normalized version of 200 rdkit descriptors
  - Disable feature scaling with flag `--no_features_scaling`.

Additional Features



- Additional machine-learning features can be added at the atom-, bond-, or molecule-level.
- **Atom-Level Features**
  - Additional atomic features can be added with argument `--atom_descriptors_path /path/to/features`.
  - Users must specify how atom descriptors are used:
    - `--atom_descriptors descriptor` concatenates new features to the embedded atomic features after the D-MPNN with an additional linear layer.
    - `--atom_descriptors feature` concatenates features to each atomic feature vector before the D-MPNN.
    - `--overwrite_default_atom_features` allows users to overwrite default atom features.
  - Atom-level descriptors and features are scaled by default. Disable scaling with flag `--no_atom_descriptor_scaling`.

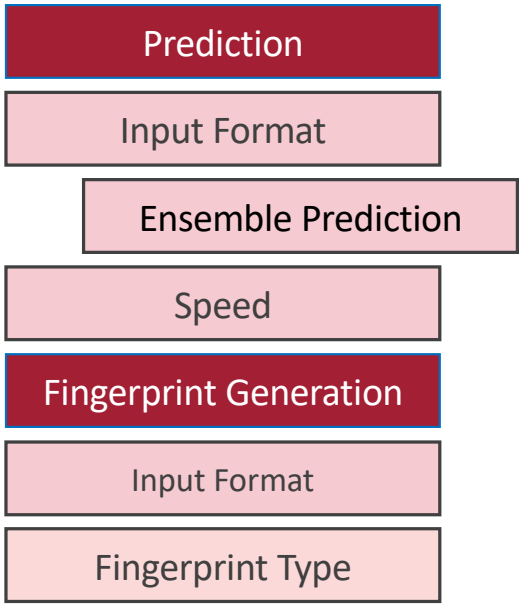
Additional Features



- Additional machine-learning features can be added at the atom-, bond-, or molecule-level.
- **Bond-Level Features**
  - Bond-level features can be added with argument `--bond_features_path /path/to/features`.
  - Bond-level features are concatenated with the bond feature vectors before the D-MPNN:
    - `--overwrite_default_bond_features` allows users to overwrite default bond features.
  - Bond-level features are scaled by default. Disable scaling with flag `--no_bond_features_scaling`.

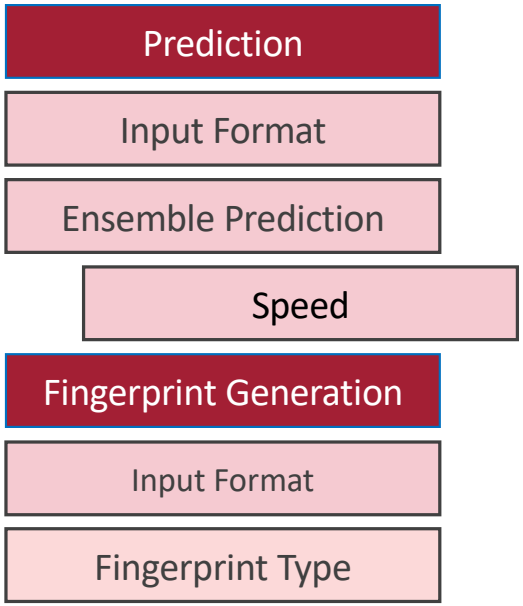
Prediction

Ensemble Prediction



- The --checkpoint_dir argument can be used to indicate a directory containing more than one model.pt file. These files can be within subdirectories.
- Predictions reported and saved will be the average of the predictions from each submodel.
- The predictions of the individual submodels can be saved by using the flag `--individual_ensemble_predictions`.
- ~~The variance among the ensemble predictions can be reported with the flag `--ensemble_variance`. This value tends to correlate with the relative uncertainty between predictions.~~

Speed of Prediction



- Relative to training, prediction is extremely fast.
- Training can take hours or days.
- Prediction with a GPU can proceed on the order of a thousand predictions per second.

Fingerprint Encoding

Input Format and Arguments

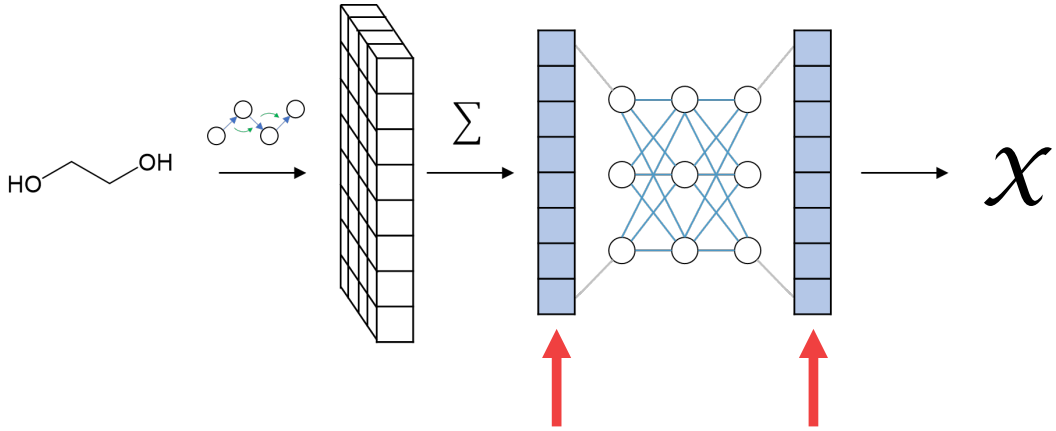
- Prediction
  - Input Format
  - Ensemble Prediction
  - Speed
- Fingerprint Generation
  - Input Format
  - Fingerprint Type

```
python fingerprint.py \
--test_path smiles/to/predict.csv \
--preds_path path/to/save.csv \
--checkpoint_dir path/to/models/dir
```

- Can be run from command line with `chemprop_fingerprint` or `python fingerprint.py`.
- Same arguments as predict.
- Required args
  - test_path – csv file with the SMILES strings to be predicted.
  - preds_path – location to write a csv file with predictions.
  - checkpoint_dir – the directory containing the model pt files

Fingerprint Type

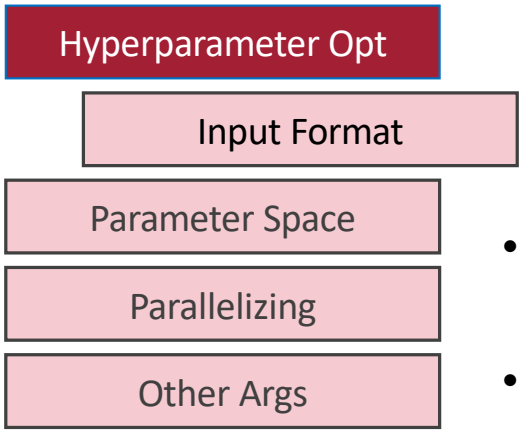
- Prediction
- Input Format
- Ensemble Prediction
- Speed
- Fingerprint Generation
- Input Format
- Fingerprint Type



- Fingerprint function returns latent representation for a given molecule, associated with a previously trained model.
- With the argument `--fingerprint_type <type>`, you can choose the location of the vector.
  - MPN – following aggregation of atomic vectors.
  - last_FFN – the output of the last FFN layer before the predicted value.
- Ensembles will return the concatenation of vectors from each model.

Hyperparameter Optimization

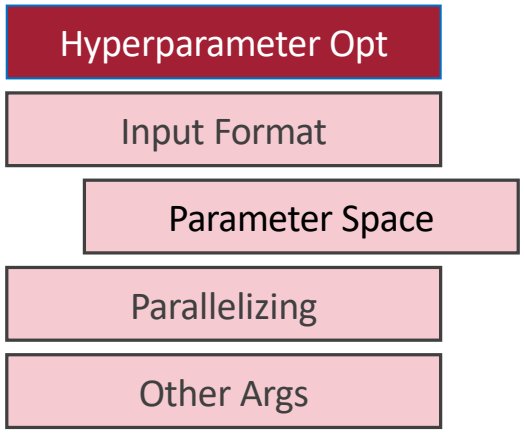
Input Format and Arguments




```
python hyperparameter_optimization.py \
--config_save_path path/to/save/args.json \
--log_dir dir/to/save/log \
--hyperopt_checkpoint_dir dir/for/checkpoints \
--num_iters 20
```

- Run using `chemprop_hyperopt` or `python hyperparameter_optimization.py`.
- Major args:
  - config_save_path – save location for the best parameter combination in json file.
  - log_dir – a log file with the result of each trial.
  - hyperopt_checkpoint_dir – the location to save checkpoints for parallelization or restarting.
  - num_iters – the number of parameter trials to run.

Parameter Search Space



These five parameters are varied over the given ranges

- depth: 2 – 5
- hidden_size: 300 – 2300
- ffn_num_layers: 1 – 3
- ffn_hidden_size: fixed same as hidden_size
- dropout: 0 – 0.4 in 0.05 increments.

Any other training arguments provided will be applied to all trials. Recommend some amount of cross-validation or ensembling, norm aggregation, and larger number of epochs.

Parallelizing

Hyperparameter Opt

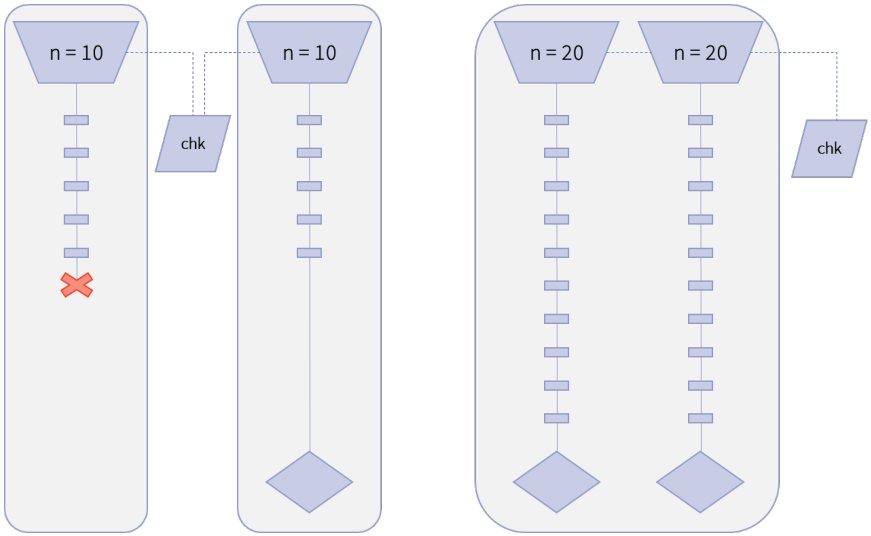
Input Format

Parameter Space

Parallelizing

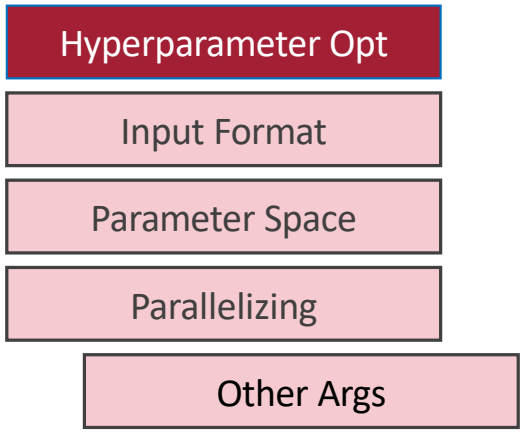
Other Args

- The checkpoint directory from --hyperopt_checkpoint_dir can be used to restart a failed optimization.
- It can also be used for parallelizing. If multiple instances of hyperparameter optimization all share the same checkpoint directory, they will share the results of trials previously run.



Recommendation: Parallelize when possible. Use it as an opportunity to run more trials. Have the instances share the same config path, log dir, and checkpoint dir. Specify separate different save_dir arguments, or don't give a save_dir at all to save considerable disk space.

Other Args – Random Steps and Manual Trials



- Initially, the search of parameter space is random.
- After a set number of trials, this process changes over to a directed search, targeting the best expected parameters.
- The number of random trials can be set with the argument `--startup_random_iters <int>`.
- Manual jobs can also be run and added to the trial results with the argument `--manual_trial_dirs <dir1> <dir2> …`
  - Be careful to match the training arguments being used with the hyperparameter search.

Chemprop in Python

Using Chemprop Within Python Scripts

To train a model, provide arguments as a list of strings (arguments are identical to command line mode), parse the arguments, and then call **chemprop.train.cross_validate():**

```
import chemprop

arguments = [
    '--data_path', 'data/tox21.csv',
    '--dataset_type', 'classification',
    '--save_dir', 'tox21_checkpoints'
]

args = chemprop.args.TrainArgs().parse_args(arguments)
mean_score, std_score = chemprop.train.cross_validate(args=args, train_func=chemprop.train.run_training)
```

Using Chemprop Within Python Scripts

For predicting with a given model, either a list of smiles or a csv file can be used as input. To use a csv file:

```
import chemprop

arguments = [
    '--test_path', 'data/tox21.csv',
    '--preds_path', 'tox21_preds.csv',
    '--checkpoint_dir', 'tox21_checkpoints'
]

args = chemprop.args.PredictArgs().parse_args(arguments)
preds = chemprop.train.make_predictions(args=args)
```

Using Chemprop Within Python Scripts

If you only want to use the predictions **preds** within the script, and not save the file, set **preds_path** to **/dev/null**. To predict on a list of smiles, run:

```
import chemprop

smiles = [['CCC'], ['CCCC'], ['OCC']]
arguments = [
    '--test_path', '/dev/null',
    '--preds_path', '/dev/null',
    '--checkpoint_dir', 'tox21_checkpoints'
]

args = chemprop.args.PredictArgs().parse_args(arguments)
preds = chemprop.train.make_predictions(args=args, smiles=smiles)
```

Discarded if a list of SMILES is provided

Using Chemprop Within Python Scripts

If you want to predict multiple sets of molecules consecutively, it is more efficient to only load the chemprop model once, and then predict with the preloaded model (instead of loading the model for every prediction):

```
import chemprop

arguments = [
    '--test_path', '/dev/null',
    '--preds_path', '/dev/null',
    '--checkpoint_dir', 'tox21_checkpoints'
]

args = chemprop.args.PredictArgs().parse_args(arguments)

model_objects = chemprop.train.load_model(args=args)

smiles = [['CCC'], ['CCCC'], ['OCC']]
preds = chemprop.train.make_predictions(args=args, smiles=smiles, model_objects=model_objects)

smiles = [['CCCC'], ['CCCCC'], ['COCC']]
preds = chemprop.train.make_predictions(args=args, smiles=smiles, model_objects=model_objects)
```

Demo - Using Chemprop Within Python Scripts

Google Colab Demo:

https://colab.research.google.com/github/chemprop/chemprop/blob/master/colab_demo.ipynb

Demo – Multiple Molecules, Multiple Targets in Bash

```
(chemprop) kpg@lambda-quad:~/scratch-chemprop$ head smiles_target_test.csv
smiles,solvent,peakwavs_max,abs_fwhm
N#Cc1cc2ccc(cc2)c2cccc3cccc(c4ccc(cc4)cc(C#N)c4ccc(cc4)c4cccc5cccc(c6ccc1cc6)c54)c32,Cc1ccccc1,350.0,51.6
CC(=O)N(Cc1ccccn1)c1c2ccccc2c2c3c(cccc13)C(=O)N(Cc1ccccn1)C2=O,CCO,437.0,70.5
Cc1ccc(-c2ccc(/C=C/c3cccn3C)cn2)nc1,ClC(Cl)Cl,380.0,67.1
c1ccc(-p2c3ccccc3c3sc4ccccc4c32)cc1,ClCCl,317.0,54.9
```

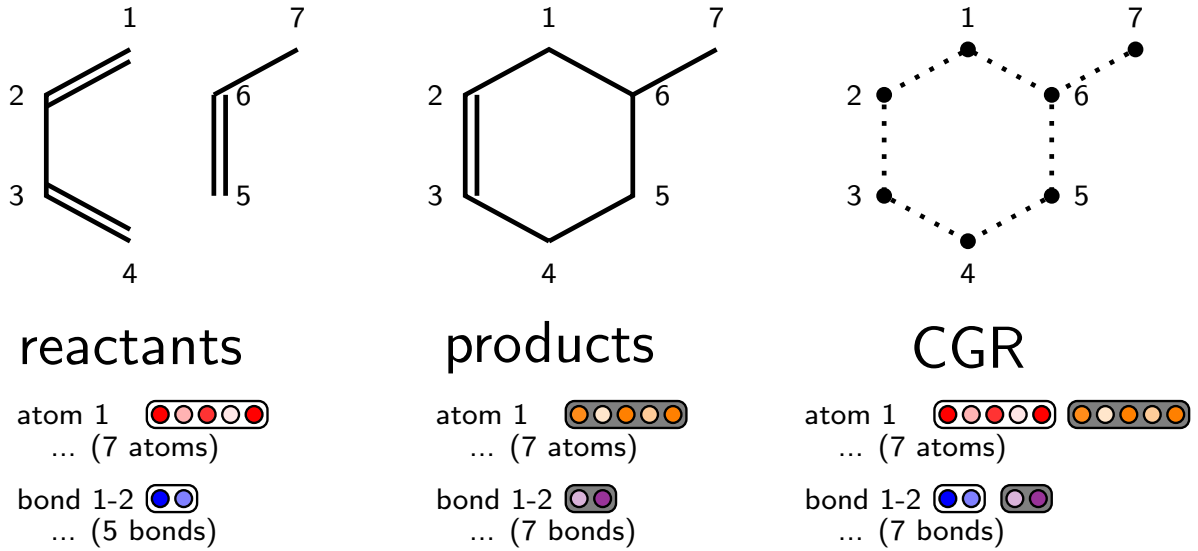
```
(base) kpg@lambda-quad:~/scratch-chemprop$ conda activate chemprop
(chemprop) kpg@lambda-quad:~/scratch-chemprop$ python ~/chemprop/train.py --data_path smiles_target_test.csv --dataset_type regression --
save_dir $(pwd) --ensemble_size 3 --number_of_molecules 2 --epochs 2 --split_key_molecule 0 --split_type scaffold_balanced --num_folds 2
```

```
(chemprop) kpg@lambda-quad:~/scratch-chemprop$ python ~/chemprop/predict.py --test_path smiles_target_test.csv --checkpoint_dir $(pwd) --
preds_path preds_unc2_2targets.csv --ensemble_variance --number_of_molecules 2
```

```
(chemprop) kpg@lambda-quad:~/scratch-chemprop$ head preds_unc2_2targets.csv
smiles,solvent,peakwavs_max,abs_fwhm,peakwavs_max_epi_unc,abs_fwhm_epi_unc
N#Cc1cc2ccc(cc2)c2cccc3cccc(c4ccc(cc4)cc(C#N)c4ccc(cc4)c4cccc5cccc(c6ccc1cc6)c54)c32,Cc1ccccc1,447.09472846306136,71.87699894809828,792.5
004704669476,47.47876612023895
CC(=O)N(Cc1ccccn1)c1c2ccccc2c2c3c(cccc13)C(=O)N(Cc1ccccn1)C2=O,CCO,419.1021356520957,70.59576975272944,527.9641977425465,19.3124867268357
97
Cc1ccc(-c2ccc(/C=C/c3cccn3C)cn2)nc1,ClC(Cl)Cl,417.71634360947064,74.23323340229274,863.2387459933626,12.772949875583512
```

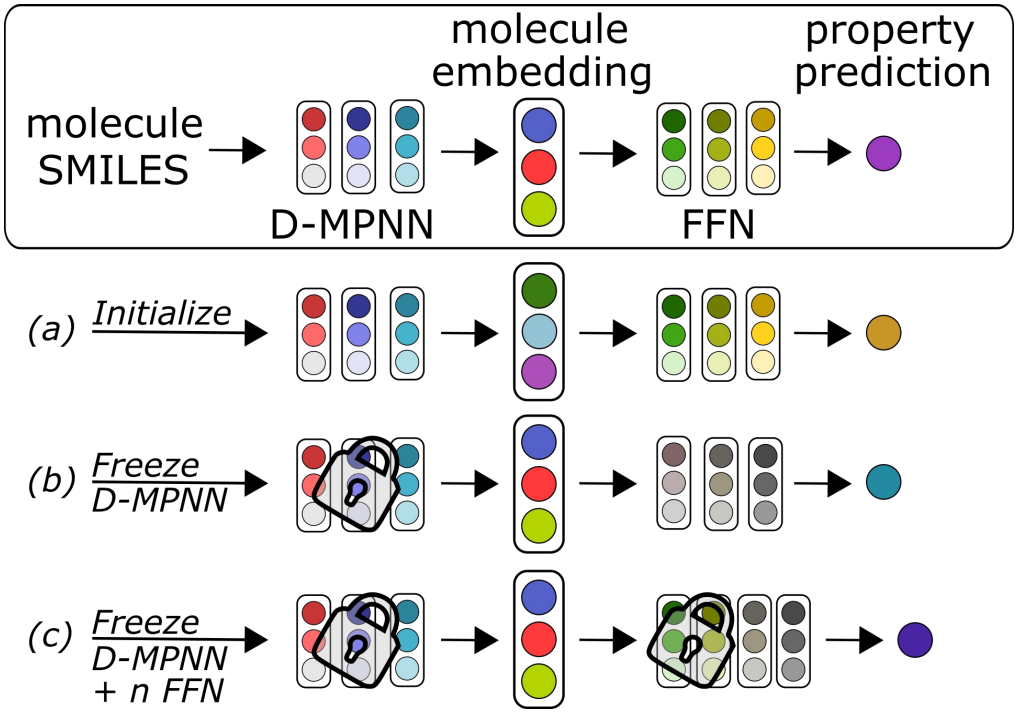
Condensed Graph of Reaction (CGR)

Reaction SMILES: `[CH2:1]=[CH1:2][CH1:3]=[CH2:4].[CH2:5]=[CH1:6][CH3:7]>>[CH2:1]1[CH1:2]=[CH1:3][CH2:4][CH2:5][CH1:6]1[CH3:7]`



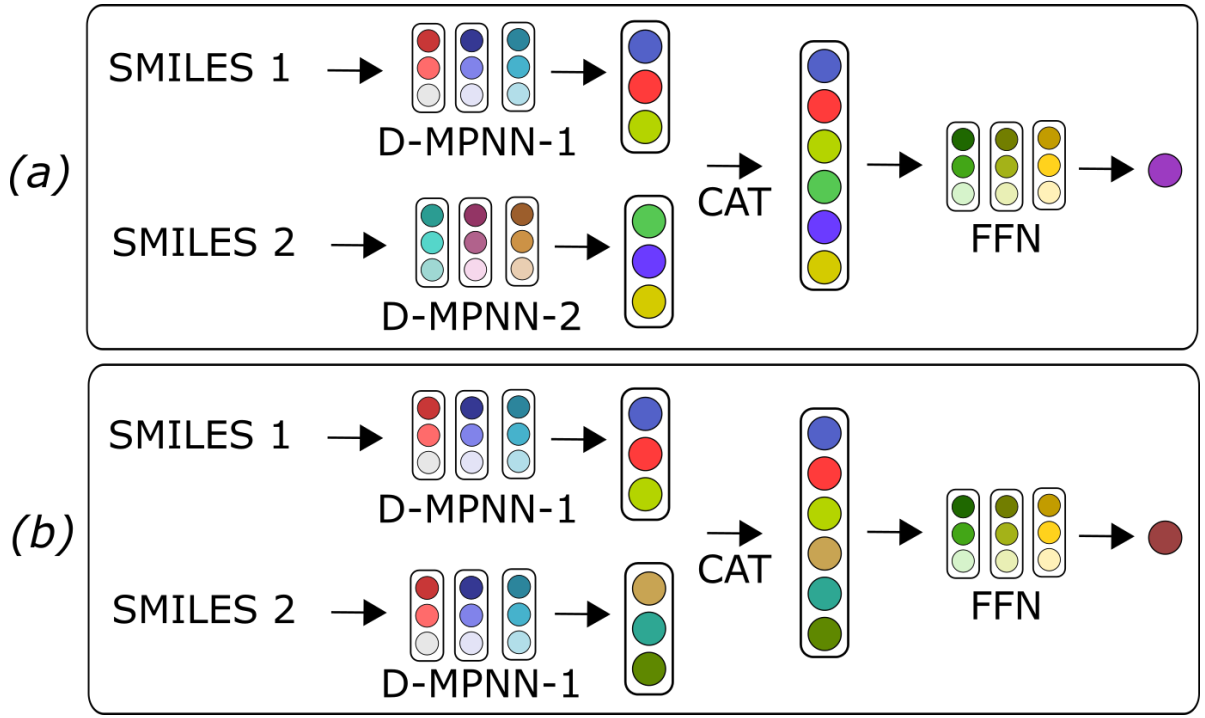
E. Heid et al. *ChemRxiv*. 2023. DOI: 10.26434/chemrxiv-2023-3zcfl

Transfer Learning



E. Heid et al. *ChemRxiv*. 2023. DOI: 10.26434/chemrxiv-2023-3zcfl

Multiple Molecules



E. Heid et al. *ChemRxiv*. 2023. DOI: 10.26434/chemrxiv-2023-3zcfl

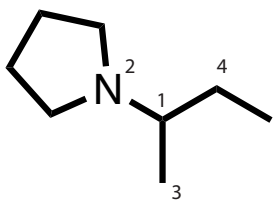
Supplementary Slides

MLPDS Steering Committee Update – July 2023

Massachusetts Institute of Technology

Chemprop learns from the molecular graph

Under the hood, chemprop utilizes a *directed* message-passing neural network (D-MPNN) to predict properties

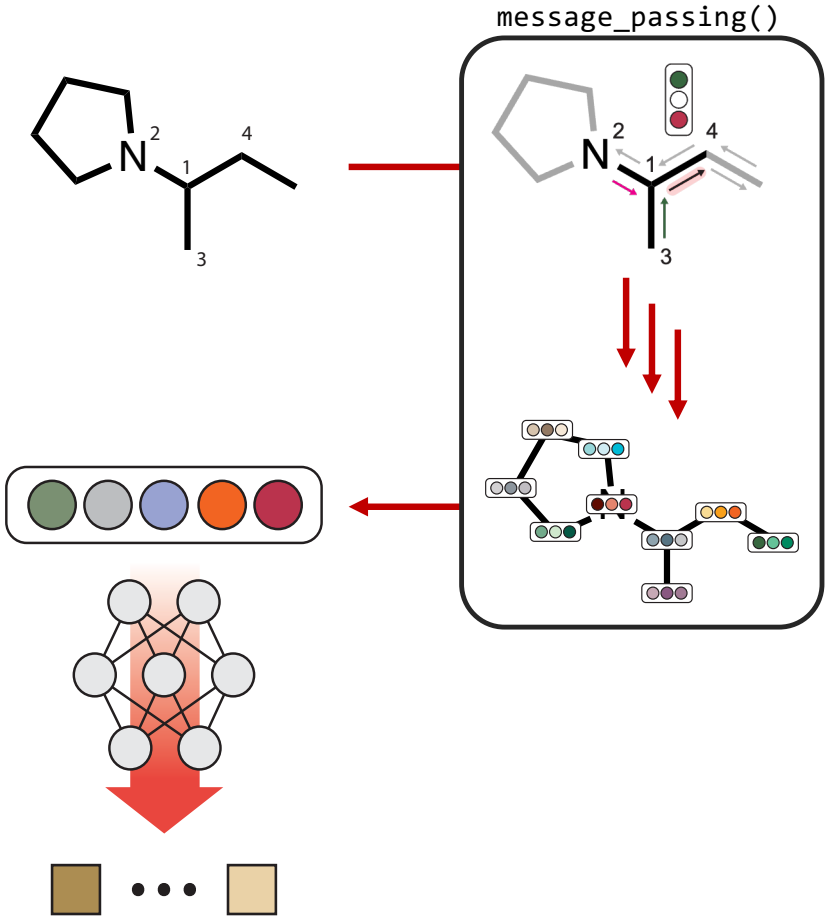


[1] *J. Chem. Inf. Model.* **2019**, *59*, 8, 3370–3388

Chemprop learns from the molecular graph

Under the hood, chemprop utilizes a *directed* message-passing neural network (D-MPNN) to predict properties

An MPNN is composed of two components:



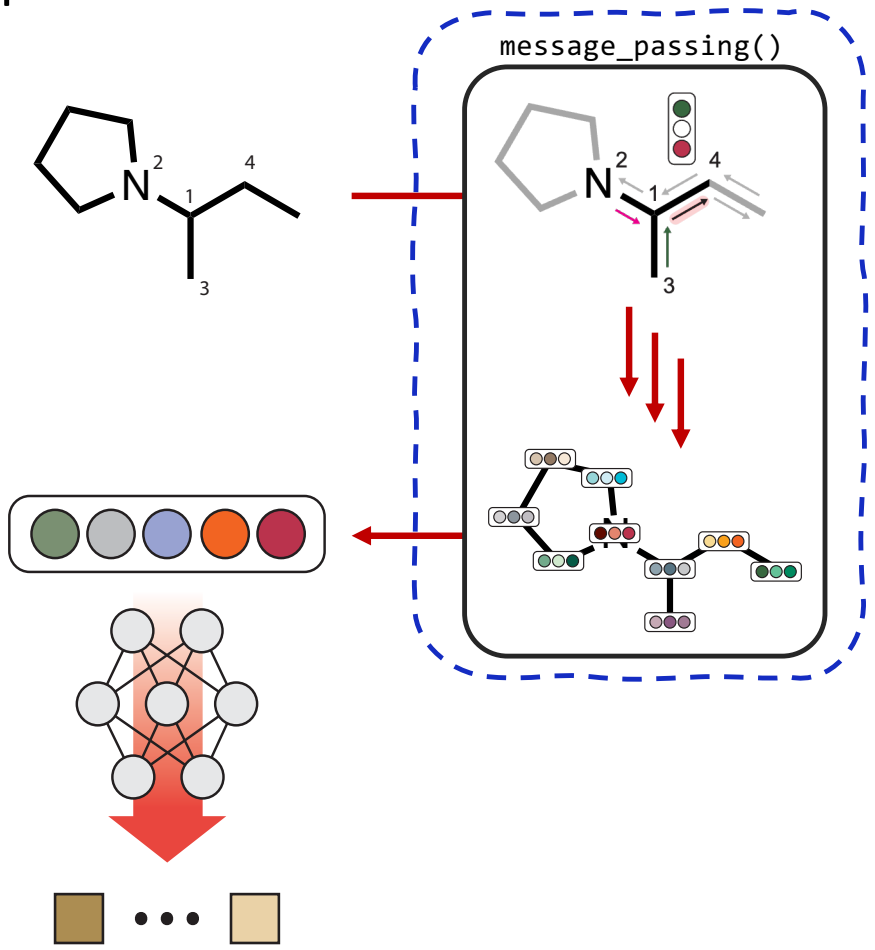
[1] *J. Chem. Inf. Model.* **2019**, *59*, 8, 3370–3388

Chemprop learns from the molecular graph

Under the hood, chemprop utilizes a *directed* message-passing neural network (D-MPNN) to predict properties

An MPNN is composed of two components:

1. message-passing layers that encode an input molecular graph



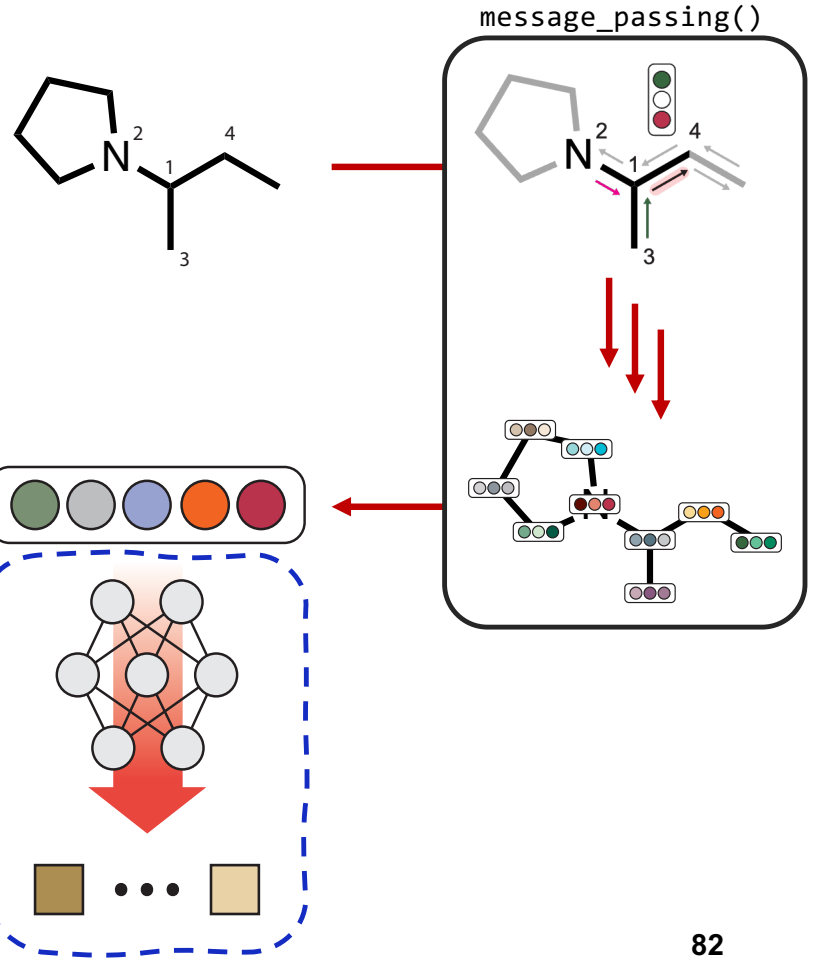
[1] *J. Chem. Inf. Model.* **2019**, *59*, 8, 3370–3388

Chemprop learns from the molecular graph

Under the hood, chemprop utilizes a *directed* message-passing neural network (D-MPNN) to predict properties

An MPNN is composed of two components:

1. message-passing layers that encode an input molecular graph
2. a feedforward neural network (FFNN) top-model that takes that encoding as input and predicts some property



[1] *J. Chem. Inf. Model.* **2019**, *59*, 8, 3370–3388

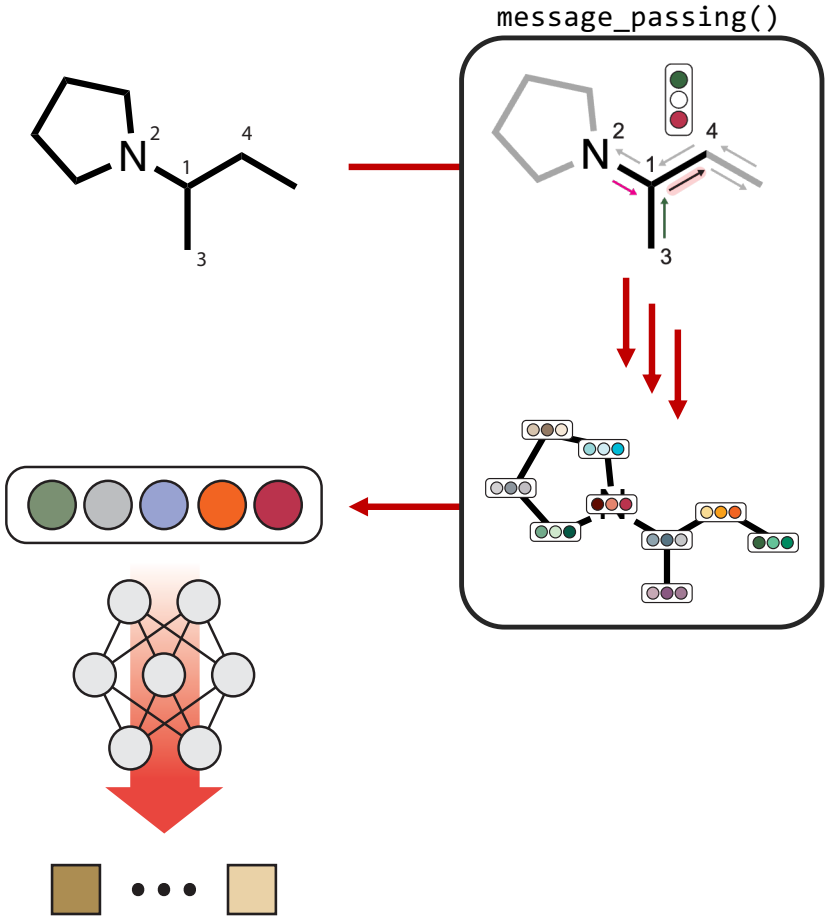
Chemprop learns from the molecular graph

Under the hood, chemprop utilizes a *directed* message-passing neural network (D-MPNN) to predict properties

An MPNN is composed of two components:

1. message-passing layers that encode an input molecular graph
2. a feedforward neural network (FFNN) top-model that takes that encoding as input and predicts some property

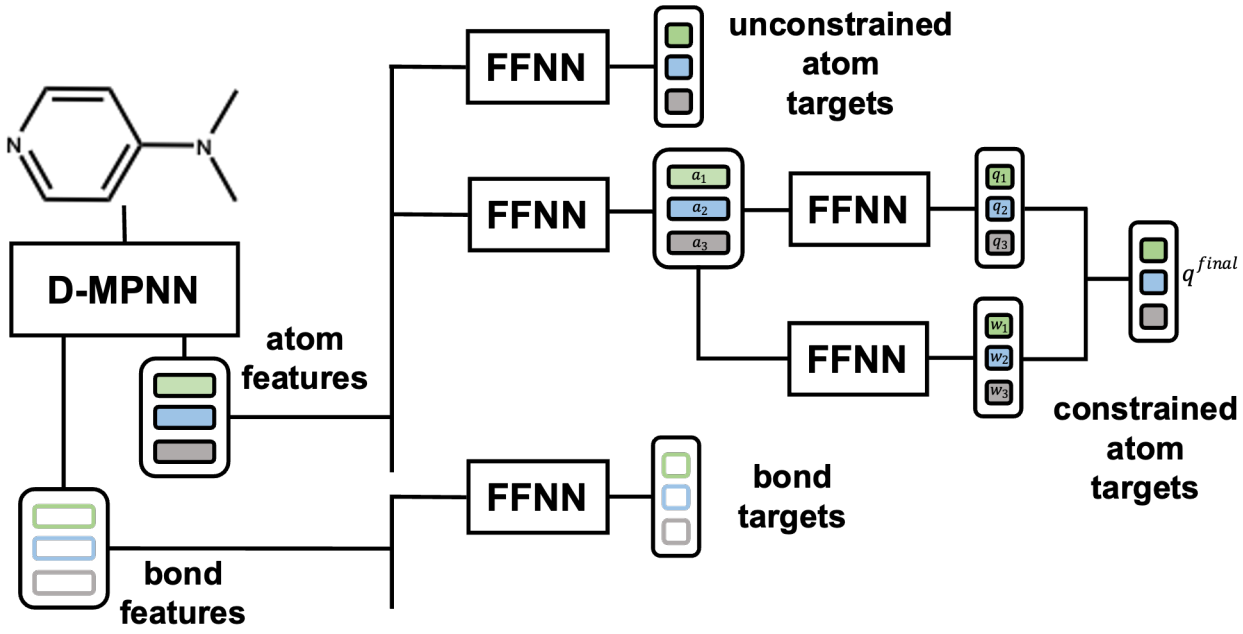
Combining (1) and (2) together enables end-to-end learning of molecular properties directly from the input molecular graph



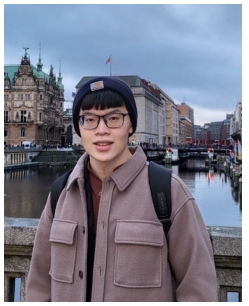
[1] *J. Chem. Inf. Model.* **2019**, *59*, 8, 3370–3388

Atom/bond-level target prediction

Multitask neural network for predicting atomic/bond descriptors with attention-based constraining method from [1]



$$q_i^{\text{final}} = q_i + \frac{w_i \left(Q - \sum_i q_i\right)}{\sum_i w_i}$$



Shih-Cheng Li

Massachusetts Institute of Technology

[1] *Chem. Sci.*, **2021**, *12*, 2198-2208

New tools for conformal uncertainty prediction

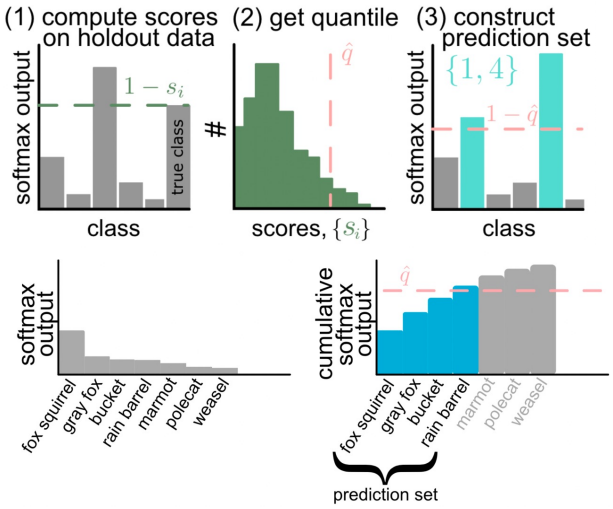
Calibration

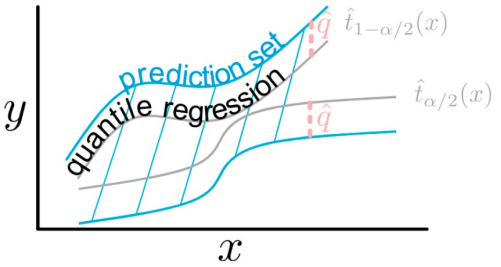
- standard conformal prediction
- adaptive conformal prediction
- conformal regression
- conformal quantile regression

Evaluation

- empirical coverage

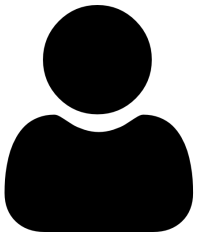
```
$ chemprop_predict \
    --calibration_method NAME \
    --conformal_alpha ALPHA \
    --evaluation_methods [METHOD ...] \
    --calibration_path PATH
```





Quantile loss (pinball loss):

$$L_\alpha = \begin{cases} (y - z)\alpha, & \text{if } y \geq z \\ (z - y)(1 - \alpha), & \text{otherwise} \end{cases}$$



Daniel Xu

Adam Fisch

[1] Angelopoulos, N., Bates, S.; arxiv:2107.07511 *[cs.LG]*, 2022