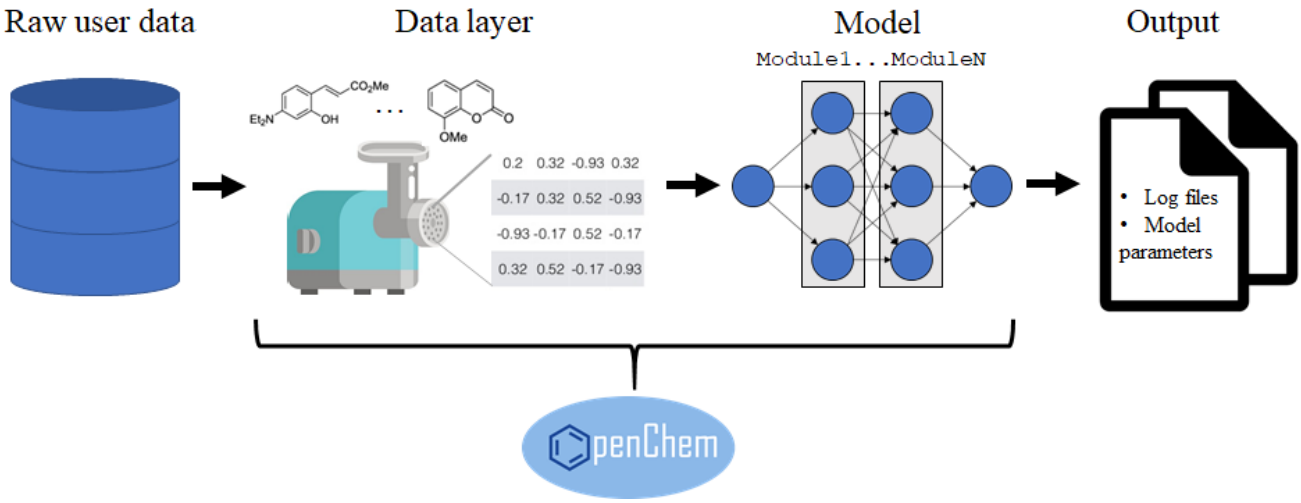
OpenChem: a deep learning toolkit for computational chemistry and drug design

Mariya Popova,[1,2] Boris Ginsburg,[3] Alexander Tropsha,[4] and Olexandr Isayev[1,2,*]*

[1] Computational Biology Department, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania
[2] Department of Chemistry, Mellon College of Science, Carnegie Mellon University, Pittsburgh, Pennsylvania
[3] NVIDIA Corporation, Santa Clara, California
[4] UNC Eshelman School of Pharmacy, University of North Carolina at Chapel Hill, Chapel Hill, North Carolina
[*] Correspondence: mpopova@andrew.cmu.edu (M.P); olexandr@olexandrisayev.com (O.I.)

Abstract: Deep learning models have demonstrated outstanding results in many data-rich areas of research, such as computer vision and natural language processing. Currently, there is a rise of deep learning in computational chemistry and materials informatics, where deep learning could be effectively applied in modeling the relationship between chemical structures and their properties. With the immense growth of chemical and materials data, deep learning models can begin to outperform conventional machine learning techniques such as random forest, support vector machines, nearest neighbor, etc. Herein, we introduce OpenChem, a PyTorch-based deep learning toolkit for computational chemistry and drug design. OpenChem offers easy and fast model development, modular software design, and several data preprocessing modules. It is freely available via the GitHub repository.



Introduction

Deep Learning is currently undergoing a rise in various fields of computational chemistry, including chemical reaction design, drug discovery, and material science. Machine learning has been a widely used technique in computational chemistry for the past 70 years, but mainly for building models for small molecule activity/property prediction from their chemical structures (or, more accurately, from chemical descriptors). Known as Quantitative Structure-Activity Relationship/Quantitative Structure-Property Relationship (QSAR/QSPR),[1] machine learning approaches have been applied to solving activity or property classification (binary, multiclass, and multilabel) as well as regression problems. Both types of problems can be solved with deep learning models as well; however, until recently, chemical data was not big enough to train robust and reliable neural networks. Currently, computational chemistry is entering the age of big data. For instance, a popular repository of chemical bioactivity data known as Pubchem [2] comprises more than 100M chemical structures, and more than 250M experimentally measured bioactivities (https://pubchemdocs.ncbi.nlm.nih.gov/statistics). Thus, the use of deep neural networks to analyze big bioactivity data becomes increasingly justified. Moreover, there are many interesting classical problems in computational chemistry that have never been tackled with machine learning before the deep learning era. An excellent example of such a problem is the *de-novo* generation of molecules with optimized properties. Previously, this problem was attacked with combinatorial methods [3, 4, 5]. However, such an approach is not very efficient since chemical space is big (with estimates up to 10^{60} molecules [6]), and an efficient sampling technique should be "smart", which makes deep learning models a good fit for this problem. Models for *de-novo* molecular design require a lot of unlabeled data, which is available at much less cost than labeled data. For example, Enamine REAL database (https://enamine.net/hit-finding/compound-collections/real-database) contains 3.7 billion of real organic molecules and can be used to train a deep generative neural network as to how to generate new realistic molecules. There have been various flavors of deep generative models for molecules in multiple representations, with most commons ones being SMILES [7] and molecular graphs.[8] There are also multiple property optimization strategies in the literature, such as reinforcement learning, Bayesian optimization, and optimization in the latent space.[9–13]

From a practical perspective, there are two main frameworks used for developing deep learning neural networks, which are PyTorch [14] and Tensorflow [15]. We decided to use PyTorch since it is more suitable for easy experimentation and fast prototyping. In other words, we wanted to build a tool that would let scientists to quickly try an idea with as little engineering efforts as possible. Here comes another pitfall for computational chemists, who, in many cases, are not computer scientists or software engineers. Even with focused deep learning libraries such as PyTorch and Tensorflow, building a deep learning network does require extensive software engineering background. There exist several community-maintained libraries for computational chemistry, such as RDKit [16] and DeepChem (https://deepchem.io). RDKit mostly offers functionality for manipulating chemical objects such as atoms, bonds, and molecules. While this functionality is extremely useful for data processing, it is not designed for building machine learning models. Another library we mentioned, DeepChem, is aimed at building deep neural networks for chemistry and built upon Tensorflow. Developing a new model with DeepChem requires writing a lot of Tensorflow code; furthermore, DeepChem does not enable modular design. Developing a new model in DeepChem would require writing code from scratch. By modular design, we mean encapsulation and reusability of standard deep neural network blocks, such as encoders, decoders, embedding layers, etc.

Another critical question is the reproducibility of computational experiments. Frequently, results reported in papers cannot be reproduced by independent researchers. This could happen due to various factors, including the absence of standardized package environments, well-tracked log files, and protocols for reproducing the results, to name a few.

To address the issues discussed above, we developed OpenChem, a deep learning library for computational chemistry built upon PyTorch framework. OpenChem offers a modular design, where

building blocks can be combined, ease of use by letting the users define a model with a single config file, and advanced deep learning features such as built-in multi-GPU and mixed-precision training support. In this paper, we introduce OpenChem design and present three case studies. All data, models, and interactive Jupyter Notebooks to reproduce these examples are available from https://github.com/Mariewelt/OpenChem.

OpenChem design

OpenChem is a deep learning toolkit for computational chemistry and drug design with PyTorch backend. The primary purpose of OpenChem is to provide computational chemists with a tool for easy experimentation with deep learning models, i.e., quick implementation of architectures, fast training, debugging, result interpretation and visualization, etc. The main idea is to implement a toolkit as a set of building blocks with a unified API that will be combined into a single custom architecture by a user.

OpenChem is implemented to offer users a modular design, i.e., blocks with the same input and output formats can be used interchangeably by changing the settings in the configuration file. For example, there are several different options for encoder block, such as RNN encoder, CNN encoder, or Graph CNN encoder, that could be used to calculate representation vectors for molecules. OpenChem allows choosing these options from the configuration file. OpenChem also supports built-in multi GPU training and offers several features for logging and debugging. Figure 1 summarizes the types of models, modules, and tasks that are currently implemented in OpenChem. Users can train predictive models for classification, regression, and multitask problems as well as develop generative models for producing novel molecular graphs with optimized properties. OpenChem can work both with SMILES string representation of molecules and molecular graphs. Data layers offer utilities for data preprocessing, such as converting SMILES strings to molecular graphs and calculating standard structural features for such graphs.

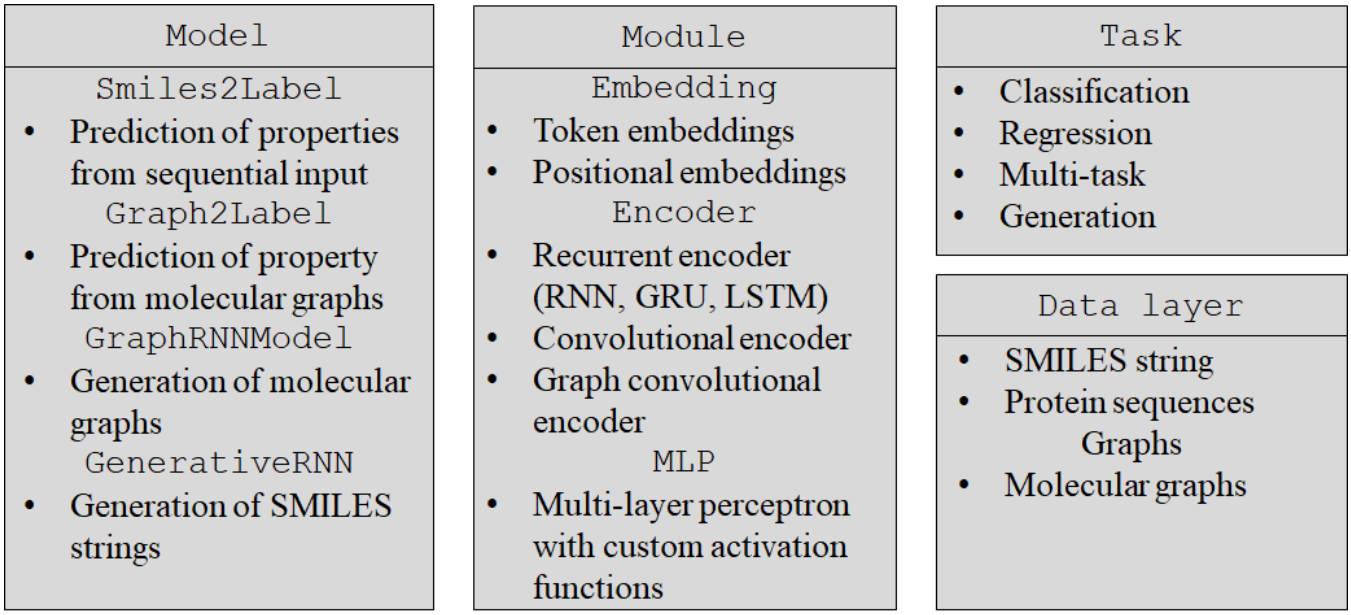


Figure 1. Main OpenChem objects

Models in OpenChem are defined in the Python configuration file as a dictionary of parameters. The dictionary must contain parameters that define how to run/train/evaluate a model as well as parameters defining model architecture. OpenChem provides scripts for model training, and it also natively supports

multi-GPU training. After the training process is finished, OpenChem will save model parameters as well as log files to a designated folder, so that the experiment can be reproduced later.

Configuration file. The configuration file must contain a model, which should be any class derived from `OpenChemModel` and dictionary `model_params`, which specifies model hyperparameters. Table 1 lists all required parameters to define a model.

Table 1. Description of standard training parameters for all models that are not related to the model architecture.

VARIABLE	DATA TYPE	DESCRIPTION	COMMENT
TASK	string	Specifies the task to be solved by the model	Could be classification, regression, multitask or generation
TRAIN_DATA_LAYER	PyTorch dataset	Contains training data	Could be None in **evaluation** mode
VAL_DATA_LAYER	PyTorch dataset	Contains validation data	Could be None in **training** mode
PRINT_EVERY	int	Specifies how often logs will be printed to stdout and log file	By default, logs are printed every epoch
SAVE_EVERY	int	Specifies how often model will be saved to checkpoints	By default model is saved every 5 epochs
LOGDIR	string	Specifies a path to the folder where model checkpoints and log files will be saved	In evaluation mode, the model will be loaded from the latest checkpoint in this directory
USE_CLIP_GRAD	boolean	Specifies if gradient clipping (CITE) will be used for training	By default, gradient clipping is not applied
MAX_GRAD_NORM	float	Specifies a maximum norm of parameters, if gradient clipping is used	N.A. if gradient clipping is not applied
BATCH_SIZE	int	Specifies the size of batches for training	Required parameter, no default value
NUM_EPOCHS	int	Specifies the number of epochs for training.	Could be None in evaluation mode
EVAL_METRICS	Python function	A user-defined function that will be used to calculate evaluation metrics.	Could be None in the training model. Majority of evaluation metrics can be found in Python scikit-learn package CITE
OPTIMIZER	PyTorch optimizer	Specifies an optimizer that will be used for training the model	Could be None in evaluation mode

Below we will consider three use cases, which will illustrate how models are defined and used in OpenChem.

Case study 1. Graph Convolution Neural Network for predicting logP

Data and model description. In this case study, we built a Graph Convolution Neural Network (GraphCNN) [17] for predicting the n-octanol/water partition coefficient (logP) directly from the chemical graphs of novel molecules. This task is an example of a regression problem, as logP covers a range of continuous values. Each molecule is represented by two matrices – adjacency matrix A and node attribute matrix H. Adjacency matrix A is a binary matrix of size $N \times N$, where N is the number of atoms in a molecule. Element A_{ij} of adjacency matrix encodes if there is an edge between atom i and atom j, i.e.:

$$A_{ij} = \begin{cases} 1, if\ there\ is\ an\ edge\ between\ i\ and\ j, \\ 0, \quad otherwise. \end{cases}$$

The second matrix is a node attributes matrix H, which has size $N \times D$, where N is again the number of atoms in a molecule, and D is the number of attributes calculated for every atom. The attributes encode the physical and chemical properties of the atoms in the context of the whole molecule.

The idea behind graph convolutions is to aggregate the information from the node attribute matrix considering the nodes adjacency, i.e., only features from the neighboring atoms will be convolved. A single layer of graph convolutions has the following form:

$$H' = \sigma(AHW),$$

where H' is the new node attribute matrix of size $N \times D'$, W is the matrix of parameters of the convolution of size $D \times D'$ and σ is a nonlinearity function such as $tanh$.

After performing a single graph convolution operation, the initial node attribute matrix H of size $N \times D$ is transformed into a new matrix H' of size $N \times D'$. In the initial matrix H row number i contains the information only about atom number i, unlike the new matrix H' where every row includes information on the corresponding atom and its neighbors.

After a single operation of graph convolution on matrix H, another operation on the node attribute matrix H'can be applied. This operation will transform matrix H' into the new node attributes' matrix H'' and further increase the receptive field [18] by aggregating information from every node, its neighbors, and also the neighbors of the neighbors. Thus, the sequential application of graph convolution operations multiple times results in an aggregation of the graph-level information in every row of the final node attribute matrix.

The final node attribute matrix is then converted into an embedding vector for the whole molecular graph using a pooling operation, such as averaging across all rows of matrix H:

$$h = \frac{1}{N}\sum_{i=1}^{N} H_i.$$

Defining model in OpenChem. Here we used five atomic properties as node attributes – atom type, valence, charge, hybridization, and aromaticity. OpenChem provides a module for declaring an attribute, where a user can specify an attribute type (node or edge), custom attribute name, attribute label (e.g., categorical), and should be one-hot encoded and list all possible values for categorical attributes. The GraphCNN model also requires a user-defined function for calculating node attributes. It is a Python function that receives the RDKit atom object as an input and returns a dictionary of atomic attributes for

the input atom. Keys in the dictionary returned by this function must match keys in the dictionary that we constructed during attributes declaration.

Next, we loaded the logP dataset (described below) into memory and split it into train and test subsets. OpenChem provides utilities for reading files. Function `read_smiles_property_file` accepts a path to the file, indices of columns that will be read from the file (indexing starts from 0), and column delimiter as arguments and returns a list where each element is a column from the file. In the example below, we loaded logP_labels.csv file from the OpenChem benchmark datasets, read columns number 1 and 2, and stored the return value of the function into a variable called data. Data is a list and its elements with indices 0 and 1 store values from columns 1 and 2 of file logP_labels.csv, respectively. Then we split the data into training and test subsets using train_test_split utility from the scikit-learn library [19] and saved the subsets into new files using save_smiles_property_file utility from OpenChem. In the next step, we create graph data layers from the saved files with train and test subsets. OpenChem provides `GraphDataset` class, which can convert SMILES strings to molecular graphs and calculate node and edge attributes. `GraphDataset` also accepts a user-defined dictionary of atomic (node) attributes, such as valency, hybridization, aromaticity, etc., and functions for computing these attributes. Other parameters to `GraphDataset` include a path to the text file with data, a list of columns that will be read from the file, and column delimiter. Once the training and test datasets were created, we specify model architecture.

Our model consists of 5 layers of Graph Convolutions with the size of a hidden layer of 128, followed by 2 layers of multilayer perceptron (MLP) with ReLU nonlinearity and hidden dimensionalities of 128 and 1. We trained the model for 100 epochs with Adam [20] optimizer with an initial learning rate of 0.01, MultiStepLR learning scheduler with step size 15, and gamma factor of 0.5, meaning that the learning rate is decreased by half every 15 epochs of training starting from the initial learning rate of 0.01. For external evaluation, we used R^2 score since the prediction of logP is a regression problem. We also wanted to see the intermediate progress report on the training and evaluation metrics printed to the standard output every 10 epochs and save the model into checkpoint every 5 epochs. We specified all these parameters in a dictionary below and saved it as python file logP_gcnn_config.py.

```
from openchem.models.Graph2Label import Graph2Label
from openchem.modules.encoders.gcn_encoder import GraphCNNEncoder
from openchem.modules.mlp.openchem_mlp import OpenChemMLP

from torch.optim import Adam
from torch.optim.lr_scheduler import StepLR
import torch.nn.functional as F
from sklearn.metrics import r2_score

model = Graph2Label

model_params = {
    'task': 'regression',
    'data_layer': GraphDataset,
    'use_clip_grad': False,
    'batch_size': 128,
    'num_epochs': 100,
    'logdir': './logs',
    'print_every': 10,
    'save_every': 5,
    'train_data_layer': train_dataset,
    'val_data_layer': test_dataset,
```

```
  'eval_metrics': r2_score,
  'criterion': nn.MSELoss(),
  'optimizer': Adam,
  'optimizer_params': {
    'lr': 0.01,
  },
  'lr_scheduler': StepLR,
  'lr_scheduler_params': {
    'step_size': 15,
    'gamma': 0.5
  },
  'encoder': GraphCNNEncoder,
  'encoder_params': {
    'input_size': train_dataset.num_features,
    'encoder_dim': 128,
    'n_layers': 5,
    'hidden_size': [128, 128, 128, 128, 128],
  },
  'mlp': OpenChemMLP,
  'mlp_params': {
    'input_size': 128,
    'n_layers': 2,
    'hidden_size': [128, 1],
    'activation': F.relu,
  }
}
```

Results. In this case study, we trained a GraphCNN for predicting the n-octanol/water partition coefficient (logP) directly from the molecular graph. The modeling dataset of 14.5K molecules was obtained based on the public version of the PHYSPROP database [21]. The dataset was curated according to our well-established protocol.[22] Structural standardization, the cleaning of salts, and the removal of mixtures, inorganics, and organometallics was performed using ChemAxon. In the case of replicate compounds, InChI Keys were generated to resolve duplicates. In case of conflicting property values, the entries were discarded. Using five-fold cross-validation, we obtained the model accuracy expressed as $R^2 = 0.90$ and root mean square error $RMSE = 0.56$. This is significantly better than traditional QSPR models ($R^2 = 0.86$ and $RMSE = 0.78$.) obtained on the same dataset using physicochemical descriptors and E-state indices [23].

Case study 2. Tox21 Challenge

Data and model description. In this case study, we built a Recurrent Neural Network (RNN) (see Fig 2). for multitask prediction of bioactivity for 12 receptors using the dataset from the Tox21 challenge [24]. This model receives a simplified molecular-input line-entry system (SMILES) string [7] for a ligand as an input and returns a vector of size 12, where each component is interpreted as a probability that the input ligand will bind to a corresponding receptor. In other words, multitask allows solving 12 independent binary classification problems with a single model. Since any SMILES string is a sequence of characters, we can use Recurrent Neural Networks to calculate a representation vector for a given molecule.[11] Each symbol of the SMILES string s_t is processed sequentially. At each time step, the model takes a single character s_t from the SMILES string, converts it to a numerical embedding vector x_t with a learnable embedding dictionary layer. Then x_t is passed to the LSTM layer:

$$h_{t+1} = W_x x_t + b_x + W_h h_t + b_h,$$

where x_t is an embedding vector for s_t obtained from the learnable embedding dictionary, and h_t is the intermediate hidden state for the SMILES prefix of length t. When the whole SMILES string is processed, hidden state h_T from the final time step is used as a representation vector for the next feed-forward layer, which outputs the vector of prediction for the input SMILES.

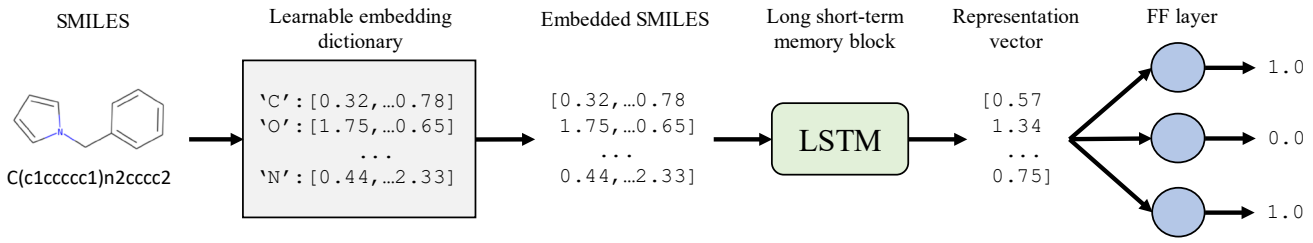


Figure 2. Scheme of multitask SMILES2Label model

Defining model in OpenChem. Tox21 dataset is available as a benchmark dataset, and it can be downloaded from the OpenChem GitHub (https://github.com/Mariewelt/OpenChem) with `read_smiles_property_file` function. Tox21 data requires some preprocessing. As it is a multi-target dataset, some of the labels are not available and, therefore, just left empty. We filled them with a dummy index, that was ignored during training. We also extracted unique tokens from the whole dataset before splitting it into train and test to avoid the situation when some of the tokens are not present in one of the pieces of the dataset. After this step, we split data randomly into training and test set and saved these subsets to new files with OpenChem `save_smiles_property_file` utility.

Next, we created the SMILES data layer from input files. We passed tokens as an argument for the data layer. We also used data augmentation by SMILES enumeration. The idea behind it is to include non-canonical notation for SMILES. Augmentation is enabled by setting the argument `augment=True` when creating an object of class `SmilesDataset`. Data augmentation is only needed for the training dataset. Since this task is multi-target, i.e., we are solving 12 separate binary classification tasks, we needed to implement a custom evaluation function for calculating classification accuracy separately for each task. As for accuracy metrics, we used the area under the receiver operation curve averaged across all classes. Next, we defined the model architecture. We used `Smiles2Label` modality. This model consists of Embedding block, Recurrent Encoder with 4 LSTM layers, and MLP. We also used dropout with a high probability to enable regularization to avoid model overfitting. For this task, we also used `MultitaskLoss` from OpenChem, which is a binary cross-entropy loss, averaged across multiple classes. Since most of the compounds in the dataset do not have labels for all 12 targets, we assigned a unique dummy label when the ground-truth class label is unknown. `MultitaskLoss` also accounts for dummy labels, and loss for these labels is not accumulated in the final loss in backpropagation.

```
model = Smiles2Label

model_params = {
  'use_cuda': True,
  'task': 'multitask',
  'use_clip_grad': True,
  'max_grad_norm': 10.0,
  'batch_size': 128,
  'num_epochs': 100,
```

```
  'logdir': './tox21_log',
  'print_every': 10,
  'save_every': 5,
  'train_data_layer': train_dataset,
  'val_data_layer': test_dataset,
  'eval_metrics': multitask_auc,
  'criterion': MultitaskLoss(ignore_index=999, n_tasks=12).cuda(),
  'optimizer': RMSprop,
  'optimizer_params': {
     'lr': 0.001,
     },
  'lr_scheduler': StepLR,
  'lr_scheduler_params': {
     'step_size': 10,
     'gamma': 0.8
  },
  'embedding': Embedding,
  'embedding_params': {
     'num_embeddings': train_dataset.num_tokens,
     'embedding_dim': 128,
     'padding_idx': train_dataset.tokens.index(' ')
  },
  'encoder': RNNEncoder,
  'encoder_params': {
     'input_size': 128,
     'layer': "LSTM",
     'encoder_dim': 128,
     'n_layers': 4,
     'dropout': 0.8,
     'is_bidirectional': False
  },
  'mlp': OpenChemMLP,
  'mlp_params': {
     'input_size': 128,
     'n_layers': 2,
     'hidden_size': [128, 12],
     'activation': F.relu,
     'dropout': 0.0
  }
}
```

Results. In this simple example, we trained a single multitask mode for predicting biological activity for 12 assays from the Tox21 challenge. Learning several tasks at the same time is performed with the aim of mutual benefits between different tasks. The similarity (and dissimilarity) between the tasks is exploited to enrich a model [25]. We obtained the mean AUC of ~0.84 with the following per target AUC values on the test set (see the benchmark dataset download page for each assay abbreviation and description in OpenChem GitHub):

- NR-AR 0.85
- NR-AR-LBD 0.90
- NR-AhR 0.87
- NR-Aromatase 0.84
- NR-ER 0.76
- NR-ER-LBD 0.82
- NR-PPAR-gamma 0.80
- SR-ARE 0.78

- SR-ATAD5 0.85
- SR-HSE 0.84
- SR-MMP 0.87
- SR-p53 0.86

These results are comparable to the results reported in the literature previously [24]. Our single multitask model approached the accuracy of the winning model, [26] which was a complex ensemble combination of different models and descriptor schemes.

Case study 3. **Generation of molecular graphs with maximized melting temperature**

Data and model description. In the final case study, we built a graph recurrent neural network MolecularRNN [27] for the generation of molecular graphs and further optimization of the specific property of the generated molecules. This model generates molecular graphs in a sequential manner, i.e., atom by atom. At each time step, the model first predicts the chemical type of a new atom and then the edge connections (including edge types) between the new atom and a previously generated one. Here we only briefly mention the overall training pipeline. Please see the full Jupyter Notebook example at https://github.com/Mariewelt/OpenChem.

Unsupervised pre-training stage. At this stage, the model is pre-trained on a vast unlabeled dataset of real molecules (such as ChEMBL or ZINC) from random initialization to produce valid and realistic molecules. In other words, during this stage, MolecularRNN learns the distribution over molecular graphs from the training dataset. So, we can assume that as the model was pretrained on a dataset of bioactive molecules from the ChEMBL database [28], the generated molecular graphs would be similar in structure and properties to known bioactive molecules.

Property optimization stage. At this stage, we use reinforcement learning with the policy gradient algorithm [29] to shift the distribution of the generated samples for some desired property that can be expressed numerically. Examples of such properties are solubility, biological activity for the desired protein, melting temperature, toxicity, etc. In this setting, the generative model is used as a policy network. We use an external predictive model to estimate the values of the desired property. This can be a machine learning model (it can be a neural network built with OpenChem), or it can be any other black box function that accepts molecule in any representation as input and output the numerical value of the desired property. According to the policy gradient algorithm, the objective function to be maximized is defined as the expected reward:

$$L(\theta) = -\sum_{i=1}^{N} r(s_N) \cdot \gamma^i \cdot log\ p(s_i|s_{i-1};\theta),$$

where s_N is the generated molecular graph, $s_i, i = 1, \ldots, N$ is the subgraph of s_N with $0 < i < N$ nodes, $\gamma$is the discount factor, $p(s_i|s_{i-1};\theta)$ is the transition probability obtained from the generative model, $r(s_N)$ is the value of the reward function for the generated molecular graph on the output of the predictive model for the desired property.

Defining the model in OpenChem. The MolecularRNN model was pretrained on the curated ChEMBL24 dataset of 1.5M molecules.[30] We used reinforcement learning to perform optimization of model parameters to maximize the melting temperature of the generated molecular graphs. The following configuration defines the MolecularRNN model, which has four GRU [31] layers with 256 hidden activations in both NodeRNN and EdgeRNN. The model learns a dictionary of embeddings for 9 atoms types (C, N, O, F, P, S, Cl, Br, I) and three bond types (single, double, and triple). We used the kekulized

form of molecules representation according to RDKit [16] which eliminated the need to specify aromatic bonds explicitly. This configuration also defines structural parameters of the molecular graphs, such as minimum and the maximum number of nodes and valency constraints for each atom type.

```
model = GraphRNNModel
model_params = {
   'task': graph_generation,
   'use_cuda': True,
   'random_seed': 0,
   'use_clip_grad': False,
   'max_grad_norm': 10.0,
   'batch_size': 1000,
   'num_epochs': 51,
   'logdir': './logs/graphrnn_log',
   'print_every': 1,
   'save_every': 1,
   'train_data_layer': train_dataset,
   'val_data_layer': test_dataset,
   'eval_metrics': get_melp_temp,
   'criterion': my_loss,
   'use_external_criterion': True,
   'optimizer': Adam,
   'optimizer_params': {
      'lr': 0.00001,
      },
   'lr_scheduler': MultiStepLR,
   'lr_scheduler_params': {
      'milestones': [100, 300, 400, 450, 480],
      'gamma': 0.3
   },
   'num_node_classes': num_node_classes,
   'num_edge_classes': num_edge_classes,
   'max_num_nodes': max_num_nodes,
   'start_node_label': start_node_label,
   'max_prev_nodes': max_pred_nodes,
   'label2atom': label2atom,
   'max_num_nodes': max_num_nodes,
   'edge2type': edge2type,
   'restrict_min_atoms': restrict_min_atoms,
   'restrict_max_atoms': restrict_max_atoms,
   'max_atom_bonds': max_atom_bonds,
   'EdgeEmbedding': Embedding,
   'edge_embedding_params': {
      'num_embeddings': num_edge_classes,
      'embedding_dim': edge_embedding_dim,
   }
   'NodeEmbedding': Embedding,
   'node_embedding_params': {
      'num_embeddings': num_node_classes,
      'embedding_dim': node_embedding_dim,
   }
   'NodeMLP': OpemChemMLPSimple,
   'node_mlp_params': {
      'input_size': 128,
      'n_layers': 2,
      'hidden_size': [128, num_node_classes],
      'activation': [nn.ReLU(inplace=True),
                     identity],
      'init': "xavier_uniform",
      }
```

```
  'NodeRNN': GRUPlain,
  'node_rnn_params': {
     'input_size': node_rnn_input_size,
     'embedding_size': 128,
     'hidden_size': 256,
     'num_layers': 4,
     'has_input': True,
     'has_output': True,
     'output_size': 128,
     'has_output_nonlin': False
     }
  'EdgeRNN': GRUPlain,
  'edge_rnn_params': {
     'input_size': edge_embedding_dim if num_edge_classes > 2 else 1,
     'embedding_size': 64,
     'hidden_size': 128,
     'num_layers': 4,
     'has_input': True,
     'has_output': True,
     'output_size': num_edge_classes if num_edge_classes > 2 else 1,
     }
  }
}
```

Results.

Currently, common modeling frameworks do not allow simultaneous ML model building, new compound generation, and property optimization. The OpenChem library bridges this gap. Briefly, to optimize melting temperature (T_{melt}), we started with our pretrained model and used the policy gradient algorithm. We trained a GraphCNN regression model to predict $T_{melt.}$ The model has RMS error of 39.5 ◦C that is comparable to the state-of-the-art for the same dataset of 54k molecules.[32] See ref [27] for complete technical details. Figure 3 shows the shift in the distribution of predicted melting temperature before and after optimization with the policy gradient algorithm, as well as examples of the generated molecules with high predicted melted temperature values.

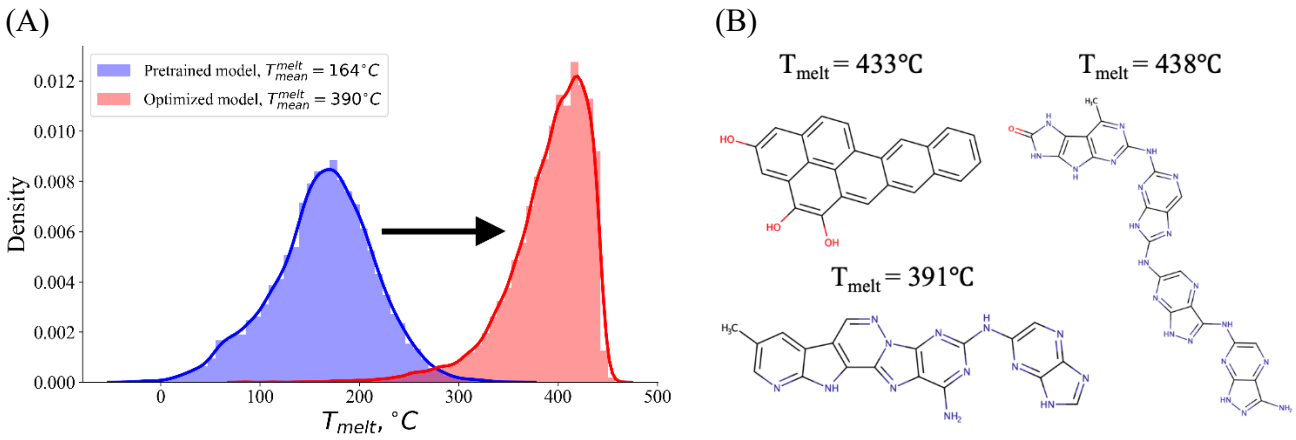


Figure 3. (A) Distribution shift in melting temperature; (B) Examples of generated molecules with high predicted melting temperatures.

Conclusions

Deep learning methods have recently emerged as a powerful approach for a variety of different tasks, including predictive, discriminative, and generative models. The OpenChem library was created to enable high-performance implementations of deep learning algorithms for drug discovery and molecular modeling applications. Built upon PyTorch framework, OpenChem is optimized for execution on GPUs and large datasets. One could quickly train ML models from datasets with hundreds of thousands or even millions of data points. OpenChem's modular API allows easy experimentation and fast model prototyping without substantial programming effort. Calculations with OpenChem could be scaled in the Cloud and HPC clusters. It provides well-tracked log files and sharable protocols and models for reproducible results. In this application note, we described just three examples of practical tasks that can be solved with OpenChem. However, the functionality of the proposed framework includes a wide variety of tasks, covering binary, multiclass, multitask classification, regression, generative modeling, and property optimization. In all three demonstrated examples, we quickly obtained a state of the art performance of models without extensive programming of each model. Our plans include extending the model list and adding new tasks such as message passing neural networks and multi-property optimization with reinforcement learning.

Acknowledgments

A.T. acknowledges NIH 1R01GM114015 and ONR N00014-16-1-2311. O.I. acknowledges support from the National Science Foundation (NSF CHE-1802789) and Eshelman Institute for Innovation (EII) award. M.P. acknowledges The Molecular Sciences Software Institute (MolSSI) Software Fellowship and NVIDIA Graduate Fellowship. We gratefully acknowledge the support and hardware donation from NVIDIA Corporation.

References:

(1) Muratov, E. N.; Bajorath, J.; Sheridan, R. P.; Tetko, I. V.; Filimonov, D.; Poroikov, V.; Oprea, T. I.; Baskin, I. I.; Varnek, A.; Roitberg, A.; Isayev, O.; Curtalolo, S.; Fourches, D.; Cohen, Y.; Aspuru-Guzik, A.; Winkler, D. A.; Agrafiotis, D.; Cherkasov, A.; Tropsha, A. QSAR without Borders. *Chem. Soc. Rev.* **2020**. https://doi.org/10.1039/d0cs00098a.

(2) Wang, Y.; Bryant, S. H.; Cheng, T.; Wang, J.; Gindulyte, A.; Shoemaker, B. A.; Thiessen, P. A.; He, S.; Zhang, J. PubChem BioAssay: 2017 Update. *Nucleic Acids Res.* **2017**, *45* (D1), D955–D963. https://doi.org/10.1093/nar/gkw1118.

(3) Cho, S. J.; Zheng, W.; Tropsha, A. Rational Combinatorial Library Design. 2. Rational Design of Targeted Combinatorial Peptide Libraries Using Chemical Similarity Probe and the Inverse QSAR Approaches. *J. Chem. Inf. Comput. Sci.* **1998**, *38* (2), 259–268. https://doi.org/10.1021/ci9700945.

(4) Gillet, V. J.; Willett, P.; Fleming, P. J.; Green, D. V. S. Designing Focused Libraries Using MoSELECT. In *Journal of Molecular Graphics and Modelling*; 2002. https://doi.org/10.1016/S1093-3263(01)00150-4.

(5) Gillet, V. J.; Khatib, W.; Willett, P.; Fleming, P. J.; Green, D. V. S. Combinatorial Library Design Using a Multiobjective Genetic Algorithm. *J. Chem. Inf. Comput. Sci.* **2002**. https://doi.org/10.1021/ci010375j.

(6) Bohacek, R. S.; McMartin, C.; Guida, W. C. The Art and Practice of Structure-Based Drug Design: A Molecular Modeling Perspective. *Medicinal Research Reviews*. 1996. https://doi.org/10.1002/(SICI)1098-1128(199601)16:1<3::AID-MED1>3.0.CO;2-6.

(7) Weininger, D. SMILES, a Chemical Language and Information System. 1. Introduction to Methodology and Encoding Rules. *J. Chem. Inf. Model.* **1988**, *28* (1), 31–36. https://doi.org/10.1021/ci00057a005.

(8) Bonchev, D.; Rouvray, D. H. *Chemical Graph Theory : Introduction and Fundamentals*; Abacus Press, 1991.
(9) Zhou, Z.; Kearnes, S.; Li, L.; Zare, R. N.; Riley, P. Optimization of Molecules via Deep Reinforcement Learning. *Sci. Rep.* **2019**. https://doi.org/10.1038/s41598-019-47148-x.
(10) Ikebata, H.; Hongo, K.; Isomura, T.; Maezono, R.; Yoshida, R. Bayesian Molecular Design with a Chemical Language Model. *J. Comput. Aided. Mol. Des.* **2017**, *31* (4), 379–391. https://doi.org/10.1007/s10822-016-0008-z.
(11) Popova, M.; Isayev, O.; Tropsha, A. Deep Reinforcement Learning for de Novo Drug Design. *Sci. Adv.* **2018**, *4* (7), eaap7885. https://doi.org/10.1126/sciadv.aap7885.
(12) Putin, E.; Asadulaev, A.; Vanhaelen, Q.; Ivanenkov, Y.; Aladinskaya, A. V.; Aliper, A.; Zhavoronkov, A. Adversarial Threshold Neural Computer for Molecular *de Novo* Design. *Mol. Pharm.* **2018**, acs.molpharmaceut.7b01137. https://doi.org/10.1021/acs.molpharmaceut.7b01137.
(13) Zhavoronkov, A.; Ivanenkov, Y. A.; Aliper, A.; Veselov, M. S.; Aladinskiy, V. A.; Aladinskaya, A. V.; Terentiev, V. A.; Polykovskiy, D. A.; Kuznetsov, M. D.; Asadulaev, A.; Volkov, Y.; Zholus, A.; Shayakhmetov, R. R.; Zhebrak, A.; Minaeva, L. I.; Zagribelnyy, B. A.; Lee, L. H.; Soll, R.; Madge, D.; Xing, L.; Guo, T.; Aspuru-Guzik, A. Deep Learning Enables Rapid Identification of Potent DDR1 Kinase Inhibitors. *Nat. Biotechnol.* **2019**, *37* (9), 1038–1040. https://doi.org/10.1038/s41587-019-0224-x.
(14) Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; Facebook, Z. D.; Research, A. I.; Lin, Z.; Desmaison, A.; Antiga, L.; Srl, O.; Lerer, A. Automatic Differentiation in PyTorch. In *Advances in Neural Information Processing Systems 32*; 2019.
(15) Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; Kudlur, M.; Levenberg, J.; Monga, R.; Moore, S.; Murray, D. G.; Steiner, B.; Tucker, P.; Vasudevan, V.; Warden, P.; Wicke, M.; Yu, Y.; Zheng, X. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016*; 2016.
(16) Landrum, G. RDkit: Open-source Cheminformatics.
(17) Kipf, T. N.; Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*; 2019.
(18) Altae-Tran, H.; Ramsundar, B.; Pappu, A. S.; Pande, V. Low Data Drug Discovery with One-Shot Learning. *ACS Cent. Sci.* **2017**, *3* (4), 283–293. https://doi.org/10.1021/acscentsci.6b00367.
(19) Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, É. Scikit-Learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**.
(20) Kingma, D. P.; Ba, J. L. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*; 2015.
(21) Beauman, J. A.; Howard, P. H. Physprop Database. *Syracuse Res. Syracuse, NY, USA* **1995**.
(22) Fourches, D.; Muratov, E.; Tropsha, A. Trust, but Verify: On the Importance of Chemical Structure Curation in Cheminformatics and QSAR Modeling Research. *J Chem Inf Model* **2010**, *50* (7), 1189–1204. https://doi.org/10.1016/j.biotechadv.2011.08.021.Secreted.
(23) Tetko, I. V.; Tanchuk, V. Y.; Villa, A. E. P. Prediction of N-Octanol/Water Partition Coefficients from PHYSPROP Database Using Artificial Neural Networks and E-State Indices. *J. Chem. Inf. Comput. Sci.* **2001**. https://doi.org/10.1021/ci010368v.
(24) Capuzzi, S. J.; Politi, R.; Isayev, O.; Farag, S. QSAR Modeling of Tox21 Challenge Stress Response and Nuclear Receptor Signaling Toxicity Assays. **2016**, *4* (February). https://doi.org/10.3389/fenvs.2016.00003.
(25) Caruana, R. Multitask Learning. *Mach. Learn.* **1997**, *28* (1), 41–75.

https://doi.org/10.1023/A:1007379606734.
(26) Mayr, A.; Klambauer, G.; Unterthiner, T.; Hochreiter, S. DeepTox : Toxicity Prediction Using Deep Learning. **2016**, *3* (February). https://doi.org/10.3389/fenvs.2015.00080.
(27) Popova, M.; Shvets, M.; Oliva, J.; Isayev, O. MolecularRNN: Generating Realistic Molecular Graphs with Optimized Properties. **2019**.
(28) Bento, A. P.; Gaulton, A.; Hersey, A.; Bellis, L. J.; Chambers, J.; Davies, M.; Krüger, F. A.; Light, Y.; Mak, L.; McGlinchey, S.; Nowotka, M.; Papadatos, G.; Santos, R.; Overington, J. P. The ChEMBL Bioactivity Database: An Update. *Nucleic Acids Res.* **2014**, *42* (D1). https://doi.org/10.1093/nar/gkt1031.
(29) Willia, R. J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.* **1992**, *8* (3), 229–256. https://doi.org/10.1023/A:1022672621406.
(30) Mendez, D.; Gaulton, A.; Bento, A. P.; Chambers, J.; De Veij, M.; Félix, E.; Magariños, M. P.; Mosquera, J. F.; Mutowo, P.; Nowotka, M.; Gordillo-Marañón, M.; Hunter, F.; Junco, L.; Mugumbate, G.; Rodriguez-Lopez, M.; Atkinson, F.; Bosc, N.; Radoux, C. J.; Segura-Cabrera, A.; Hersey, A.; Leach, A. R. ChEMBL: Towards Direct Deposition of Bioassay Data. *Nucleic Acids Res.* **2019**. https://doi.org/10.1093/nar/gky1075.
(31) Chung, J.; Gulcehre, C.; Cho, K.; Bengio, Y. Gated Feedback Recurrent Neural Networks. *arXiv* **2015**.
(32) Tetko, I. V; Sushko, Y.; Novotarskyi, S.; Patiny, L.; Kondratov, I.; Petrenko, A. E.; Charochkina, L.; Asiri, A. M. How Accurately Can We Predict the Melting Points of Drug-like Compounds? *J. Chem. Inf. Model.* **2014**, *54* (12), 3320–3329. https://doi.org/10.1021/ci5005288.