Advanced Graph and Sequence Neural Networks for Molecular Property Prediction and Drug Discovery

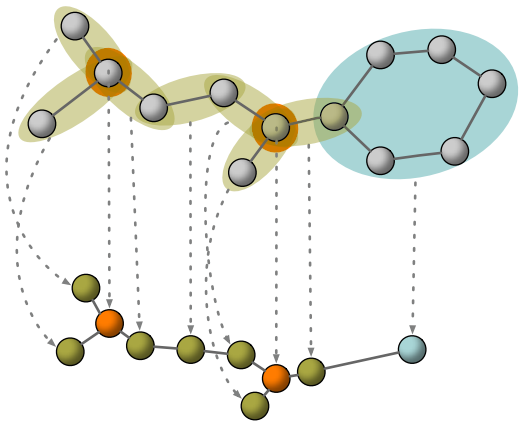
Zhengyang Wang[1,+], Meng Liu[1,+], Youzhi Luo[1,+], Zhao Xu[1,+], Yaochen Xie[1,+], Limei Wang[1,+], Lei Cai[1,2+], Qi Qi[3], Zhuoning Yuan[3], Tianbao Yang[3], and Shuiwang Ji[1,*]

[1]Texas A&M University, Department of Computer Science and Engineering, College Station, TX 77843, USA
[2]Currently with Microsoft, Bellevue, WA 98004, USA
[3]University of Iowa, Department of Computer Science, Iowa City, IA 52242, USA
[*]Correspondence should be addressed to: sji@tamu.edu
[+]These authors contributed equally to this work.

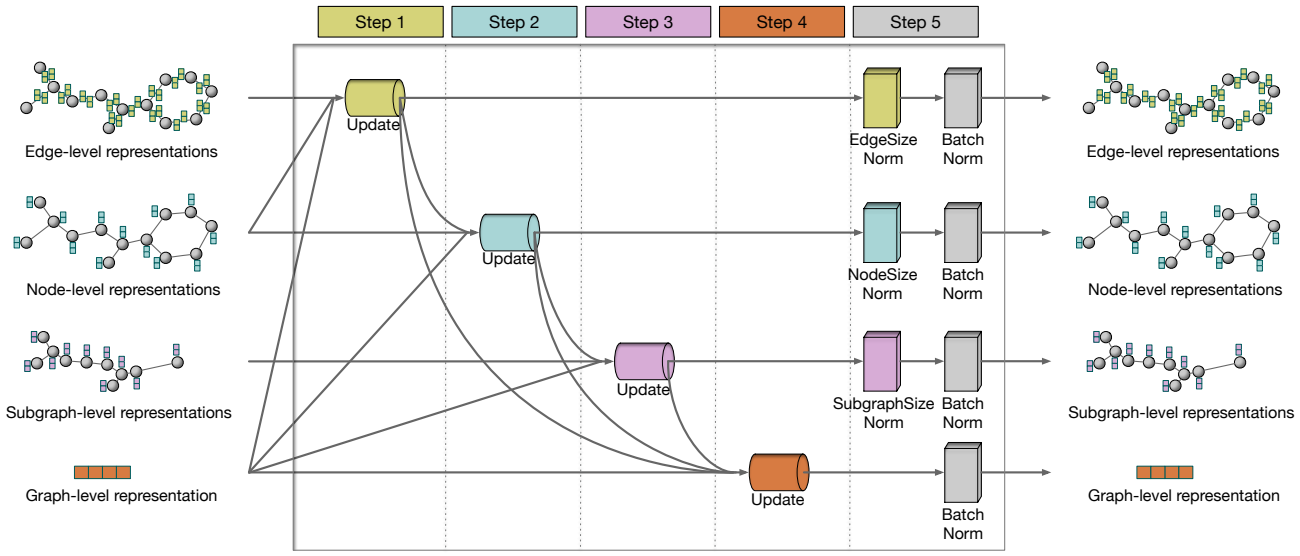
List of Figures

List of Tables

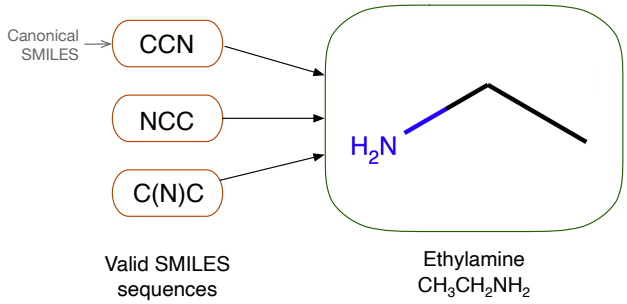
Contents



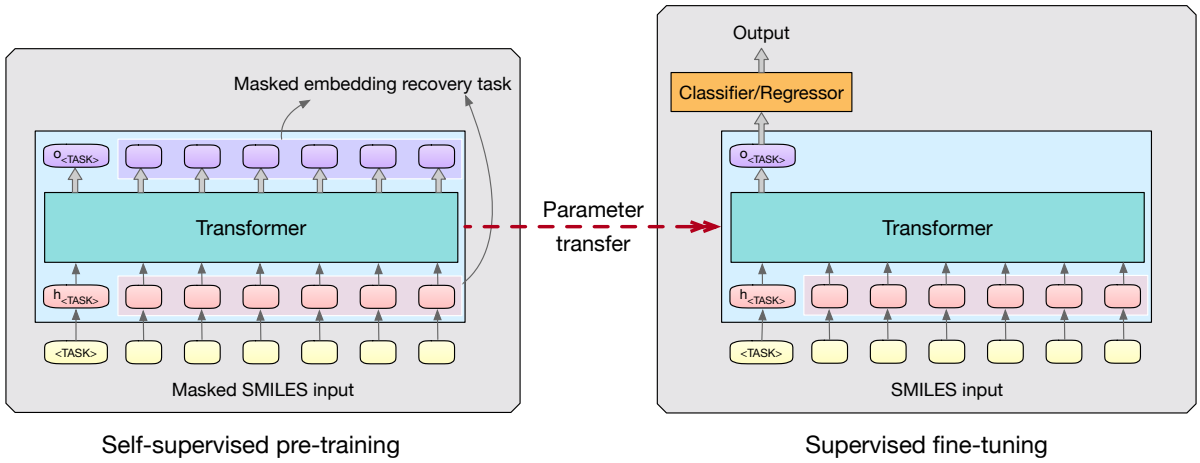
Supplementary Figure 1. An example of obtaining the junction tree from a molecular graph. ■, ■, and ■ represent singletons, bonds, and rings, respectively.



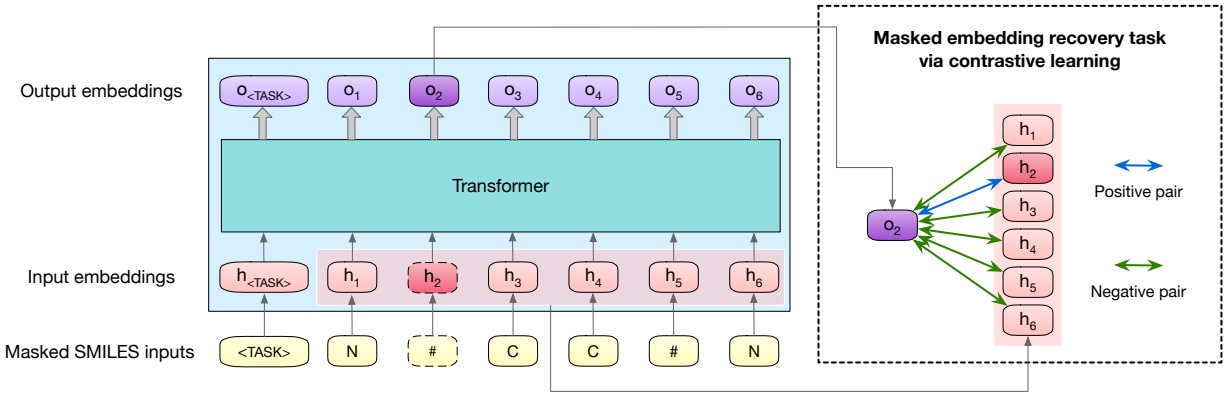
Supplementary Figure 2. An illustration of a single ML-MPNN layer. ML-MPNN performs message passing hierarchically among multi-level representations to obtain the updated multi-level representations.



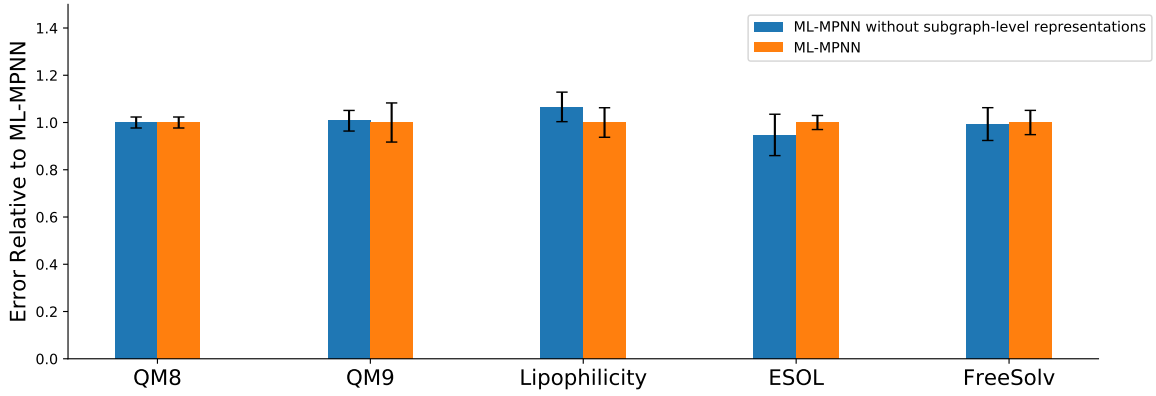
Supplementary Figure 3. An example of multiple valid SMILES sequences for a molecule. Among all valid SMILES sequences, *CCN* is the canonical SMILES of ethylamine.



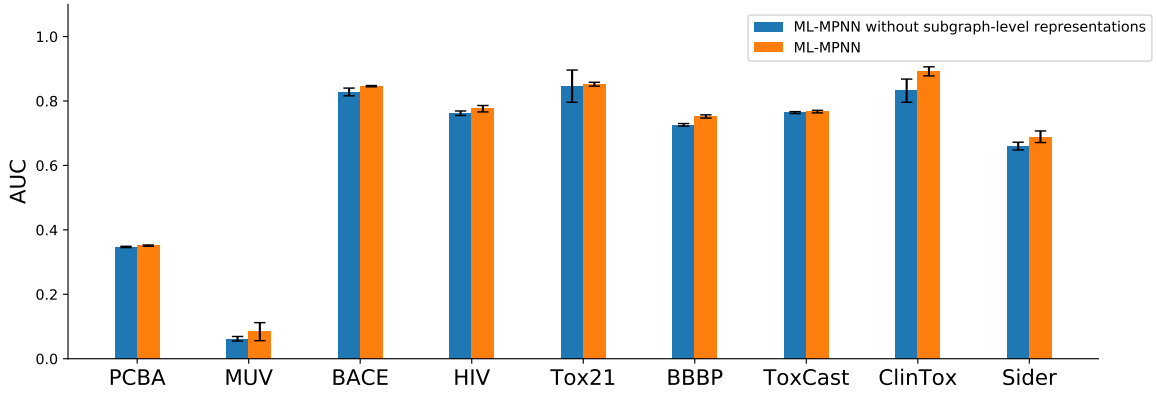
Supplementary Figure 4. An illustration of Contrastive-BERT. The contrastive-BERT is pre-trained via a masked embedding recovery task, and then be fine-tuned on downstream molecular property prediction tasks.



Supplementary Figure 5. An illustration of the pre-training phase of contrastive-BERT. The self-supervised masked embedding recovery task uses contrastive learning to encourage the similarity of positive pairs and dissimilarity of negative embedding pairs simultaneously.

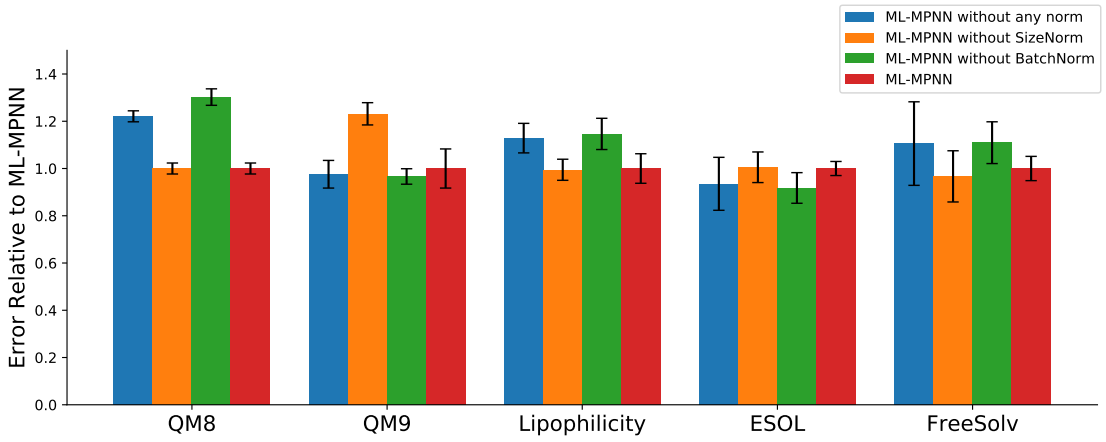


(a) Performance on regression datasets (lower is better)

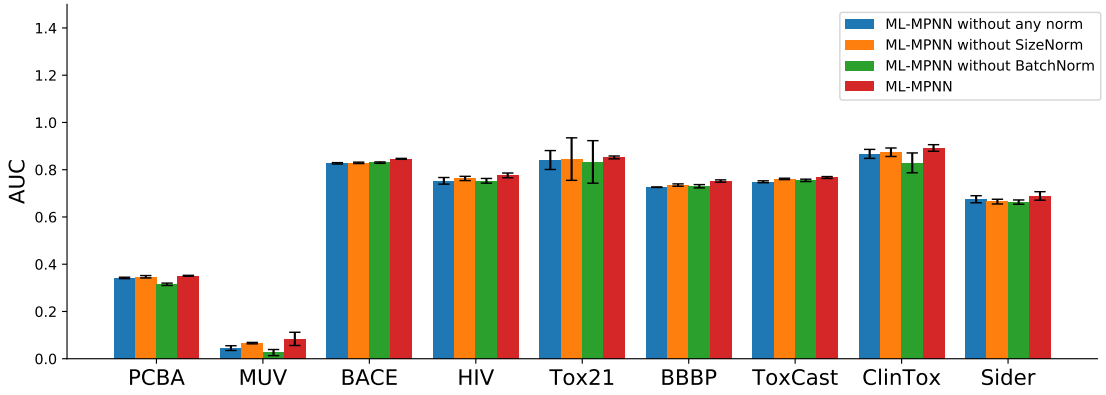


(b) Performance on classification datasets (higher is better)

Supplementary Figure 6. Comparisons of prediction performance between ML-MPNN and ML-MPNN without subgraph-level representations. a, Prediction performances of ML-MPNN and ML-MPNN without subgraph-level representations on the testing dataset of 5 regression tasks, in terms of MAE or RMSE (Section 4.5, Supplementary Table 1). **b**, Prediction performance of ML-MPNN and ML-MPNN without subgraph-level representations on the testing dataset of 9 classification tasks, in terms of ROC-AUC or PRC-AUC (Section 4.5, Supplementary Table 1). All the figures follow the same settings as Fig. 2b&c. All the corresponding quantitative results are provided in Supplementary Table 3. These results show the significance of subgraph-level representations.

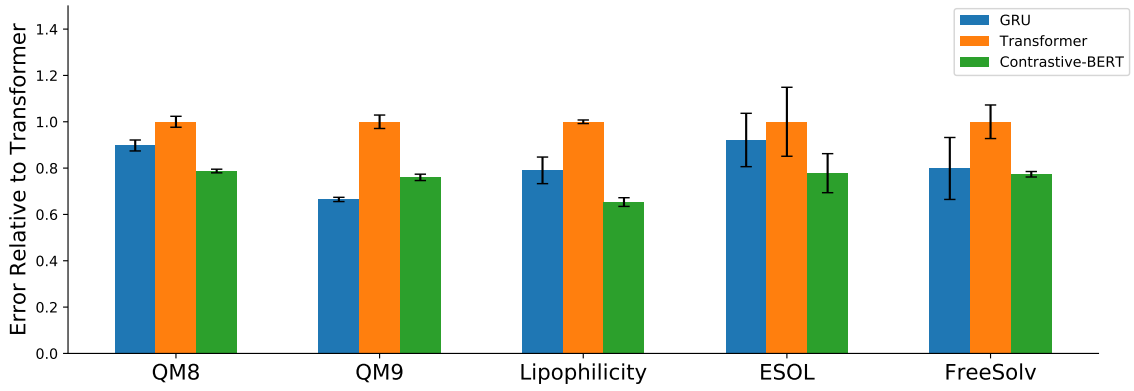


(a) Performance on regression datasets (lower is better)

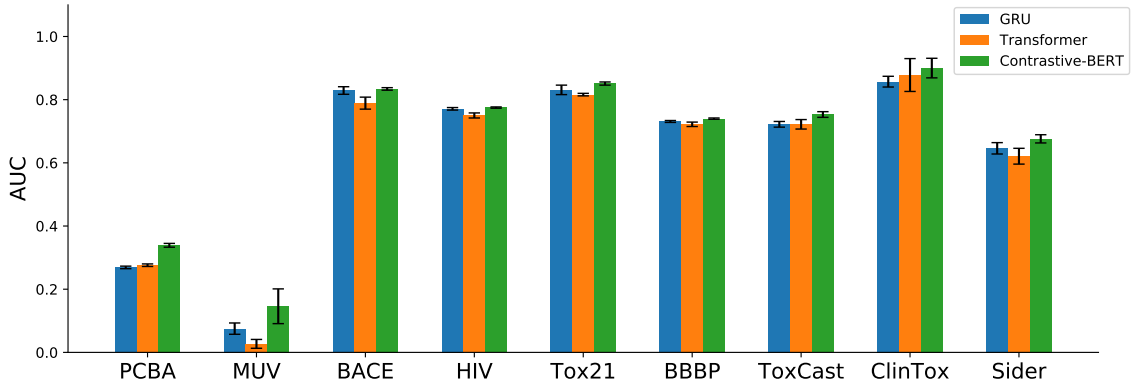


(b) Performance on classification datasets (higher is better)

Supplementary Figure 7. Comparisons of prediction performance between ML-MPNN and ML-MPNN without normalization for representations. a, Prediction performances of ML-MPNN and ML-MPNN without normalization for representations on the testing dataset of 5 regression tasks, in terms of MAE or RMSE (Section 4.5, Supplementary Table 1). **b**, Prediction performance of ML-MPNN and ML-MPNN without normalization for representations on the testing dataset of 9 classification tasks, in terms of ROC-AUC or PRC-AUC (Section 4.5, Supplementary Table 1). All the figures follow the same settings as Fig. 2b&c. All the corresponding quantitative results are provided in Supplementary Table 4. These results demonstrate the effectiveness of the normalization techniques.

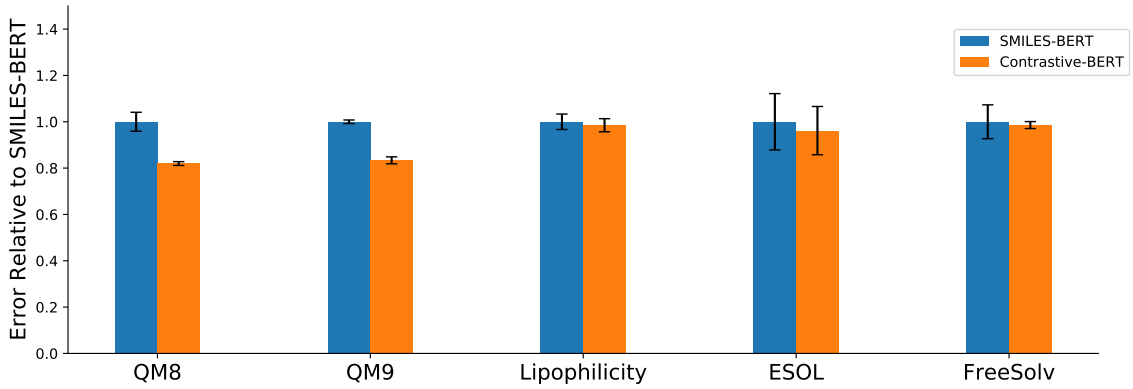


(a) Performance on regression datasets (lower is better)

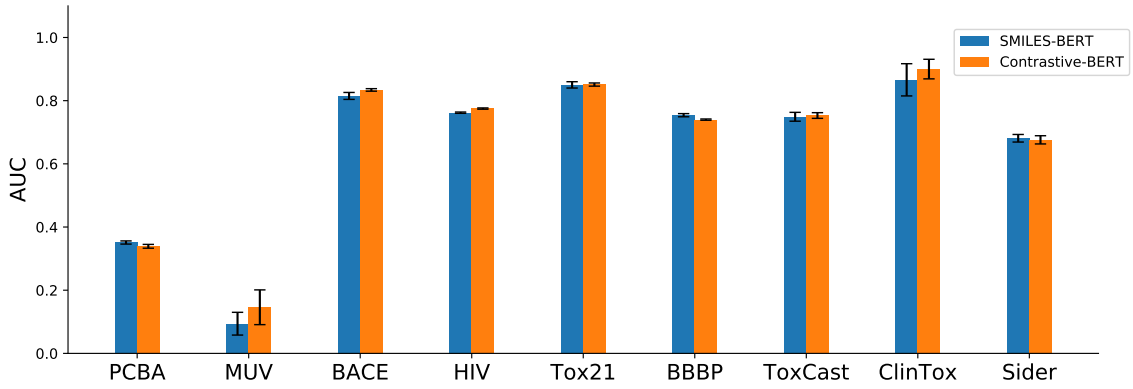


(b) Performance on classification datasets (higher is better)

Supplementary Figure 8. Comparisons of prediction performance between contrastive-BERT, GRU[1] and Transformer[2]. a, Prediction performances of contrastive-BERT, GRU and Transformer on the testing dataset of 5 regression tasks, in terms of MAE or RMSE (Section 4.5, Supplementary Table 1). The error relative to the Transformer baseline is plotted. **b**, Prediction performance of contrastive-BERT, GRU and Transformer on the testing dataset of 9 classification tasks, in terms of ROC-AUC or PRC-AUC (Section 4.5, Supplementary Table 1). The absolute performance values are plotted. All the figures follow the same settings as Fig. 2b&c. All the corresponding quantitative results are provided in Supplementary Table 5. These results demonstrate the necessity of effective pre-training.



(a) Performance on regression datasets (lower is better)



(b) Performance on classification datasets (higher is better)

Supplementary Figure 9. Comparisons of prediction performance between contrastive-BERT and SMILES-BERT[3]. **a**, Prediction performances of contrastive-BERT and SMILES-BERT on the testing dataset of 5 regression tasks, in terms of MAE or RMSE (Section 4.5, Supplementary Table 1). The error relative to the SMILES-BERT baseline is plotted. **b**, Prediction performance of contrastive-BERT and SMILES-BERT on the testing dataset of 9 classification tasks, in terms of ROC-AUC or PRC-AUC (Section 4.5, Supplementary Table 1). The absolute performance values are plotted. All the figures follow the same settings as Fig. 2b&c. All the corresponding quantitative results are provided in Supplementary Table 6. These results demonstrate the advantages of our proposed masked embedding recovery task.

Dataset	# Tasks	Task type	# Molecules	Split	Metric
QM8	12	Regression	21786	Random	MAE
QM9	12	Regression	133885	Random	MAE
ESOL	1	Regression	1128	Random	RMSE
FreeSolv	1	Regression	643	Random	RMSE
Lipophilicity	1	Regression	4200	Random	RMSE
PCBA	128	Classification	439863	Random	PRC-AUC
MUV	17	Classification	93127	Random	PRC-AUC
HIV	1	Classification	41913	Scaffold	ROC-AUC
BACE	1	Classification	1522	Scaffold	ROC-AUC
BBBP	1	Classification	2042	Scaffold	ROC-AUC
Tox21	12	Classification	8014	Random	ROC-AUC
ToxCast	617	Classification	8589	Random	ROC-AUC
SIDER	27	Classification	1427	Random	ROC-AUC
ClinTox	2	Classification	1485	Random	ROC-AUC

Supplementary Table 1. Summary of the 14 datasets from MoleculeNet[4] used in our experiments.

Dataset	Metric	MoleculeNet	D-MPNN	Contrastive-BERT	ML-MPNN	AdvProp
QM8	MAE	0.0143±0.001	0.011±0.000	0.0100±0.0001	0.0086±0.0002	**0.0085±0.0001**
QM9	MAE	2.4±1.1	2.666± 0.006	2.489±0.045	2.017±0.167	**1.920±0.061**
Lipophilicity	RMSE	0.655± 0.036	0.555± 0.023	0.592±0.017	0.560±0.035	**0.515±0.025**
ESOL	RMSE	0.58±0.03	0.555± 0.047	0.554±0.060	0.571±0.017	**0.513±0.036**
FreeSolv	RMSE	1.15± 0.12	1.075± 0.054	1.174±0.018	1.052±0.054	**0.990±0.035**
PCBA	PRC-AUC	0.136±0.004	0.335±0.001	0.339±0.006	0.351±0.002	**0.382±0.005**
MUV	PRC-AUC	**0.184±0.020**	0.041±0.007	0.146±0.055	0.084±0.028	0.134±0.020
BACE	ROC-AUC	0.867±0.008	-	0.834±0.004	0.846±0.002	**0.871±0.002**
HIV	ROC-AUC	0.792±0.000	0.776±0.008	0.775±0.002	0.776±0.001	**0.795±0.006**
Tox21	ROC-AUC	0.829±0.006	0.851±0.002	0.851±0.005	0.852±0.006	**0.868±0.008**
BBBP	ROC-AUC	0.729±0.000	0.738±0.001	0.740±0.002	0.752±0.005	**0.772±0.001**
ToxCast	ROC-AUC	0.742±0.003	-	0.753±0.009	0.767±0.004	**0.779±0.001**
ClinTox	ROC-AUC	0.832±0.037	0.864±0.017	0.900±0.031	0.892±0.014	**0.962±0.018**
Sider	ROC-AUC	0.684±0.009	0.676±0.014	0.676±0.013	0.689±0.018	**0.718±0.022**

Supplementary Table 2. Prediction performances of contrastive-BERT, ML-MPNN and AdvProp on MoleculeNet benchmarks. The mean and standard deviation results are computed from 3 independent runs over 3 different train/validation/test splits on each dataset, which are the same splits as those in Fig. 2b&c.

Dataset	Metric	ML-MPNN without subgraph-level representations	ML-MPNN
QM8	MAE	**0.0086±0.0002**	**0.0086±0.0002**
QM9	MAE	2.032±0.088	**2.017±0.167**
Lipophilicity	RMSE	0.597±0.035	**0.560±0.035**
ESOL	RMSE	**0.541±0.05**	0.571±0.017
FreeSolv	RMSE	**1.045±0.073**	1.052±0.054
PCBA	PRC-AUC	0.347±0.002	**0.351±0.002**
MUV	PRC-AUC	0.062±0.007	**0.084±0.028**
BACE	ROC-AUC	0.828±0.012	**0.846±0.002**
HIV	ROC-AUC	0.762±0.007	**0.776±0.01**
Tox21	ROC-AUC	0.846±0.005	**0.852±0.006**
BBBP	ROC-AUC	0.726±0.004	**0.752±0.005**
ToxCast	ROC-AUC	0.764±0.003	**0.767±0.004**
ClinTox	ROC-AUC	0.832±0.036	**0.892±0.014**
Sider	ROC-AUC	0.660±0.012	**0.689±0.018**

Supplementary Table 3. Comparisons of prediction performance between ML-MPNN and ML-MPNN without subgraph-level representations. The mean and standard deviation results are computed from 3 independent runs over 3 different train/validation/test splits on each dataset, which are the same splits as those in Fig. 2b&c.

Dataset	Metric	ML-MPNN without any norm	ML-MPNN without SizeNorm	ML-MPNN without BatchNorm	ML-MPNN
QM8	MAE	0.0105±0.0002	**0.0086±0.0002**	0.0112±0.0003	**0.0086±0.0002**
QM9	MAE	1.968±0.118	2.484±0.095	**1.949±0.066**	2.017±0.167
Lipophilicity	RMSE	0.632±0.035	0.557±0.025	0.642±0.037	**0.560±0.035**
ESOL	RMSE	0.534±0.064	0.574±0.037	**0.524±0.037**	0.571±0.017
FreeSolv	RMSE	1.163±0.186	**1.017±0.114**	1.167±0.093	1.052±0.054
PCBA	PRC-AUC	0.342±0.003	0.347±0.005	0.315±0.005	**0.351±0.002**
MUV	PRC-AUC	0.045±0.010	0.066±0.003	0.026±0.013	**0.084±0.028**
BACE	ROC-AUC	0.827±0.003	0.829±0.003	0.830±0.003	**0.846±0.002**
HIV	ROC-AUC	0.753±0.014	0.763±0.009	0.753±0.010	**0.776±0.01**
Tox21	ROC-AUC	0.841±0.004	0.845±0.009	0.833±0.009	**0.852±0.006**
BBBP	ROC-AUC	0.726±0.001	0.735±0.005	0.730±0.007	**0.752±0.005**
ToxCast	ROC-AUC	0.749±0.004	0.761±0.003	0.755±0.005	**0.767±0.004**
ClinTox	ROC-AUC	0.867±0.019	0.874±0.018	0.829±0.042	**0.892±0.014**
Sider	ROC-AUC	0.675±0.015	0.665±0.010	0.663±0.009	**0.689±0.018**

Supplementary Table 4. Comparisons of prediction performance between ML-MPNN and ML-MPNN without normalization for representations. The mean and standard deviation results are computed from 3 independent runs over 3 different train/validation/test splits on each dataset, which are the same splits as those in Fig. 2b&c.

Dataset	Metric	GRU	Transformer	Contrastive-BERT
QM8	MAE	0.0114±0.0003	0.0127±0.0003	**0.0100±0.0001**
QM9	MAE	**2.177± 0.030**	3.275±0.095	2.489±0.045
Lipophilicity	RMSE	0.716±0.052	0.906±0.007	**0.592±0.017**
ESOL	RMSE	0.656±0.082	0.712±0.106	**0.554±0.060**
FreeSolv	RMSE	1.212±0.203	1.518±0.110	**1.174±0.018**
PCBA	PRC-AUC	0.269±0.004	0.276±0.004	**0.339±0.006**
MUV	PRC-AUC	0.0752±0.0179	0.027±0.014	**0.146±0.055**
BACE	ROC-AUC	0.829±0.012	0.789±0.019	**0.834±0.004**
HIV	ROC-AUC	0.771±0.004	0.750±0.008	**0.775±0.002**
Tox21	ROC-AUC	0.831±0.015	0.816±0.004	**0.851±0.005**
BBBP	ROC-AUC	0.731±0.003	0.722±0.007	**0.740±0.002**
ToxCast	ROC-AUC	0.722±0.009	0.722±0.015	**0.753±0.009**
ClinTox	ROC-AUC	0.857±0.017	0.878±0.052	**0.900±0.031**
Sider	ROC-AUC	0.646±0.018	0.621±0.025	**0.676±0.013**

Supplementary Table 5. Comparisons of prediction performance between contrastive-BERT, GRU[1] and Transformer[2]. The mean and standard deviation results are computed from 3 independent runs over 3 different train/validation/test splits on each dataset, which are the same splits as those in Fig. 2b&c.

Dataset	Metric	SMILES-BERT	Contrastive-BERT
QM8	MAE	0.0122±0.0005	**0.0100±0.0001**
QM9	MAE	2.986± 0.023	**2.489±0.045**
Lipophilicity	RMSE	0.601± 0.020	**0.592±0.017**
ESOL	RMSE	0.576± 0.070	**0.554±0.060**
FreeSolv	RMSE	1.191± 0.087	**1.174±0.018**
PCBA	PRC-AUC	**0.351±0.005**	0.339±0.006
MUV	PRC-AUC	0.094±0.036	**0.146±0.055**
BACE	ROC-AUC	0.815± 0.011	**0.834±0.004**
HIV	ROC-AUC	0.762±0.002	**0.775±0.002**
Tox21	ROC-AUC	0.85±0.01	**0.851±0.005**
BBBP	ROC-AUC	**0.754±0.005**	0.740±0.002
ToxCast	ROC-AUC	0.749±0.014	**0.753±0.009**
ClinTox	ROC-AUC	0.866±0.051	**0.900±0.031**
Sider	ROC-AUC	**0.681±0.012**	0.676±0.013

Supplementary Table 6. Comparisons of prediction performance between contrastive-BERT and SMILES-BERT[3]. The mean and standard deviation results are computed from 3 independent runs over 3 different train/validation/test splits on each dataset, which are the same splits as those in Fig. 2b&c.

Dataset	Initial learning rate	Learning rate decay factor	Weight decay	Hidden dimension	Dropout rate	Batch size
QM8	0.0001	0.8	0	256	0	64
QM9	0.0001	0.8	0.00005	256	0	64
Lipophilicity	0.0005	0.5	0	128	0	64
ESOL	0.0005	0.8	0	128	0	64
FreeSolv	0.001	0.8	0	128	0	32
PCBA	0.0005	0.5	0.00005	256	0	1024
MUV	0.0005	0.5	0.00005	32	0	64
BACE	0.001	0.5	0.00005	32	0.3	64
HIV	0.0005	0.5	0.00005	32	0	64
Tox21	0.0005	0.5	0.0005	128	0	64
BBBP	0.0005	0.5	0.00005	32	0.3	64
ToxCast	0.0005	0.5	0	64	0	64
ClinTox	0.0005	0.5	0.0005	256	0	64
Sider	0.0005	0.5	0.00005	128	0	64
DRH[5]	0.0025	0.5	0	64	0.2	64
AI Cures[6]	0.0005	0.5	0.00005	128	0.7	64

Supplementary Table 7. Task-specific settings of hyperparameters in ML-MPNN.

Dataset	Batch size	# Epochs	Learning rate
QM8	128	300	0.00005
QM9	128	300	0.00002
Lipophilicity	128	100	0.00002
ESOL	32	100	0.00002
FreeSolv	32	200	0.00002
PCBA	128	100	0.00002
MUV	128	50	0.00002
BACE	128	100	0.00002
HIV	64	30	0.00002
Tox21	128	50	0.00002
BBBP	128	50	0.00001
ToxCast	128	100	0.00002
ClinTox	64	50	0.00002
Sider	32	100	0.00002
DRH[5]	128	100	0.000025
AI Cures[6]	64	100	0.00002

Supplementary Table 8. Task-specific Settings of hyperparameters in contrastive-BERT during the fine-tuning phase.

Dataset	# Iterations	Subsequence length	Decay factor	Normalization
QM8	1	7	1.0	Yes
Lipophilicity	5	6	1.0	Yes
ESOL	2	10	1.0	No
FreeSolv	3	4	1.0	No
BACE	5	5	0.8	Yes
HIV	7	5	0.8	No
Tox21	11	15	0.8	Yes
BBBP	7	7	0.7	No
ClinTox	3	3	1.0	Yes
Sider	6	5	0.7	Yes
DRH[5]	7	11	0.8	No
AI Cures[6]	5	11	0.8	Yes

Supplementary Table 9. Task-specific settings of hyperparameters in the two kernel methods.

1 Supplementary Note: Software usage instructions

Our code is available at Github. You can use the following code to download and install it.

```
$ git clone https://github.com/divelab/MoleculeX.git
$ cd MoleculeX/AdvProp
```

The use of AdvProp requires the running of three models (sequence-based, graph-based and kernel-based) with four output results. The four output results are then ensembled as the final prediction. The environment requirements for the three models might have conflict and we hence recommend create individual environments and run the results of the three models following the instructions in this section.

Before executing the scripts for the first time for each model, execute the follow command for each model.

```
$ cd [sequence/graph/kernels]
$ chmod +x scripts/*.sh
```

1.1 Usage instructions for sequence-based methods

1.1.1 Environment Setup

The required core libraries includes *numpy*, *scikit-learn*, *pytorch (v1.4.0)*, *rdkit (v2018.09.1)*, and *LibAUC*[7]. We highly recommend setup the required environment with Anaconda by executing the following commands to install and activate the environment:

```
$ conda env create -f sequence.yaml
$ source activate sequence
```

Note that the cudatoolkit version is 10.0.130 in sequence.yaml. If the cuda version is different in your machine, you may need to change the cudatoolkit version in sequence.yaml before installing it.

1.1.2 Reproduce our results with our trained model on MoleculeNet

Download our trained models from https://drive.google.com/drive/u/1/folders/1mmYvDaYLnAwACNS52rVaBkmIlUgBHEmc. Specify gpu id, dataset name, seed for random split (122, 123, 124) and model path in `scripts/run_reproduce.sh`. Then execute

```
$ bash scripts/run_reproduce.sh
```

1.1.3 Use MoleculeNet for training and prediction on your own dataset

I. Pre-training

In this code, we implement two pretrain tasks for downstream molecule property prediction task:

- Mask prediction task, that is predicting the ids of masked tokens in a sequence.
- Mask contrastive learning task is our propsed pretrain task, that is predicting the output embedding of masked tokens in a sequence, as described in our paper.

You can download our provided pretrained model from https://drive.google.com/drive/folders/1auvkx5e-3OI9kUeH8CjVm8e9R1kLgz5H?usp=sharing. These models are trained on selected 2M molecules from ZINC dataset, using mask prediction task and mask contrastive learning task separately.

To do pretraining by yourself, you can firstly modify the configuration file `/config/pretrain_config.py` following the instruction and then execute the this command

```
$ bash scripts/run_pretrain.sh
```

II. Training and predicting

To do molecule property prediction on your own dataset, firstly download our provided pretrained model from https://drive.google.com/drive/folders/1auvkx5e-3OI9kUeH8CjVm8e9R1kLgz5H?usp=sharing, then modify the configuration file `/config/train_config.py` following the instruction. Execute different scripts for different ways to split training, validation and test set.

- Case 1: Using split data files (train, validation, test)

  If you provide three `.csv` data files for training, validation and test datasets separately, then specify the path to your training data file, the path to your validation data file, gpu ids and the directory to store your output results in `scripts/run_train1.sh`, and execute it

  ```
  $ bash scripts/run_train1.sh
  ```

  After training finalized, specify the path to your test data file, gpu ids and the path to your stored model file in `scripts/run_evaluate1.sh`, and execute it

  ```
  $ bash scripts/run_evaluate1.sh
  ```

  If you do not have labels for test data, you can run `scripts/run_predict1.sh` for prediction. The prediction result will be saved as a `prediction.npy` file in your specified directory.

  ```
  bash scripts/run_predict1.sh
  ```

- Case 2: Using a single data file

  If you solely provide a .csv data file containing training, validation and test datasets all together , you need to choose one split method from our implemented three split methods (random split, stratified split or scaffold split, refer to https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5868307/ for details). Specify the path to your data file, gpu ids, split method, split ratio and split seed in `scripts/run_train2.sh`, and execute

```
$ bash scripts/run_train2.sh
$ bash scripts/run_evaluate2.sh
```

for prediction and evaluation. For prediction only, replace `run_evaluate2.sh` by `run_predict2.sh` in the above command.

During training, a folder with the same name as the out variable in the training script will be created, and all the output results (model parameters as .pth files, validation result recorded in record.txt, etc.) will be automatically saved under this folder.

To reproduce our trained models of MoleculeNet datasets, you can copy the content of the corresponding configuration file under the `config/MoleculeNet_config` folder into `config/train_config.py`.

Note that your data file must be csv files, where one column with the key 'smiles' stores all smile strings and the other columns store property.

1.2 Usage instructions for graph-based methods

1.2.1 Environment Setup

- If you use CUDA10.0, you can easily setup the environment using the provided yaml file. Please make sure Anaconda is installed firstly. Then you can execute the following commands one by one to install and activate the environment.

```
$ conda env create -f graph.yaml
$ source activate graph  (or conda activate graph)
$ pip install torch-scatter==latest+cu100 -f \
    https://pytorch-geometric.com/whl/torch-1.4.0.html
$ pip install torch-sparse==latest+cu100 -f \
    https://pytorch-geometric.com/whl/torch-1.4.0.html
$ pip install torch-cluster==latest+cu100 -f \
    https://pytorch-geometric.com/whl/torch-1.4.0.html
$ pip install torch-spline-conv==latest+cu100 -f \
    https://pytorch-geometric.com/whl/torch-1.4.0.html
$ pip install torch-geometric
$ pip install git+https://github.com/bp-kelley/descriptastorus
$ pip install pandas_flavor
```

- If you use other CUDA version, you can setup the environment manually. The versions of key packages we used are listed as follows.

  - PyTorch v1.4.0

- PyTorch Geometric v1.6.0
- RDKit v2020.03.3
- LibAUC[7]

Note that the versions of PyTorch and PyTorch Geometric should be compatible and PyTorch Geometric is related to other packages, which should be compatible with your CUDA version. It would be easy to install PyTorch Geometric correctly by following the https://pytorch-geometric.readthedocs.io/en/latest/notes/installation.html#.

1.2.2 Reproduce our results with our trained model on MoleculeNet

I. Convert original SMILES string to Pytorch Geometric Data type. An example is avaliable in `scripts/tran_data.sh`. You can execute the command

```
$ bash ./scripts/tran_data.sh
```

II-a. To train our model from scratch, execute

```
$ bash ./scripts/train.sh
```

Otherwise, follow the next step to use our trained model.

II-b. Load our trained model and do prediction. An example is avaliable in `scripts/predict.sh`. You can execute the command

```
$ bash ./scripts/predict.sh
```

1.2.3 Use MoleculeNet for training and prediction on your own dataset

To run your own datasets, you need to firstly create a `config_YourDatasetName.py` under `./config/`. A default configuration can be found at `./config/config_default.py`. Then follow the steps (II.1) and (II.2a).

1.3 Usage instructions for kernel methods

1.3.1 Environment Setup

The required core libraries includes *descriptastorus*, *grake*, *numpy*, *pandas_flavor*, *python (≤ v3.7.10)*, *rdkit*, *scikit-learn (≤ v0.21.3)* and *shogun*. If you have installed Anaconda, you can execute the following commands to install and activate the environment:

```
$ conda env create -f kernel.yaml
$ source activate kernels
$ pip install git+https://github.com/bp-kelley/descriptastorus
```

1.3.2 Reproduce our results with our trained model on MoleculeNet

We provide scripts and configurations for the datasets used in MoleculeNet. For example, run the command below to train and evaluate the model with graph kernel on Freesolv dataset with seeds 122, 123 and 124.

```
$ bash scripts/freesolv_graphkernel.sh
```

Run the command below to train and evaluate the model with sequence kernel on clintox with seeds 122, 123 and 124.

```
$ bash scripts/clintox_seqkernel.sh
```

We also provide the trained models for the datasets. To use the trained models and skip training, insert the argument `-mode test` or `-mode predict` after `python ../src/main.py` in the scripts.

1.3.3 Use MoleculeNet for training and prediction on your own dataset

Use the following commands,

```
$ python ../src/main.py --mode train_eval [--args]
$ python ../src/main.py --mode train [--args]
$ python ../src/main.py --mode test [--args]
$ python ../src/main.py --mode predict [--args]
```

to train, evaluate and predict with your own datasets, where `[-args]` are the additional arguments you need to specify. If an argument is not specified, the default setting will be used.

I. Explaination of the four modes

- `train_eval`: will run training (with trained model saved to an `.pkl` file) and the evaluation (with prediction saved to an `.npy` file) after training. If a single data file is provided, will split the data for training and evaluation.
- `train`: will run training (with trained model saved to an `.pkl` file only. If a single data file is provided, all data in the file will be used for training.
- `test`: will load the tained model and run prediction (saved to an `.npy` file) and evaluation. If a single data file is provided, will split for testing; if splitted `.csv` files are provided, all data in the test file will be used.
- `predict`: will load the tained model and run prediction (saved to an `.npy` file). If a single data file is provided, will split for testing; if splitted `.csv` files are provided, all data in the test file will be used.

II. Arguments for dataset loading

With a single `[dataset_name].csv` file with all the data for training and testing: in this case, you need to specify the arguments

- `-dataset`: the name of your dataset, should be the same to the name of the data file and the key of your config,

- `-data_path`: name of the folder where you put your dataset (`.csv`) files,
- `-seed`: the seed for randomly split the train/test data (default: 122),
- `-split_mode`: the split mode: random, stratified or scaffold (default: random),
- `-split_train_ratio` and `-split_valid_ratio`: the ratios of training examples and validation examples (default: 0.8/0.1).

Then the train-test splitting will be based on the above arguments you provided.

With splitted `.csv` files for train/val/test: in this case, you need to include `-split_ready` and specify the arguments

- `-dataset`: the name of your dataset, should be the same to the key of your config,
- `-trainfile`: path to the `.csv` file that stores the training examples,
- `-validfile`: path to the `.csv` file that stores the validation examples,
- `-testfile`: path to the `.csv` file that stores the testing examples.

Note that all the `.csv` files must include the "smiles" column and can contain any number of columns as the labels (ground truths).

III. Argument for storing your results.

You can specify the path where you store your trained models and the predictions by specifying

- `-model_path`: path to the folder where the trained model will be stored,
- `-prediction_path`: path to the folder where the prediction results will be stored.

The trained model will be automatically named as `[dataset_name]_[seed].pkl` and the prediction result will be named as `[dataset_name]_seed_[seed].npy`.

IV. Other arguments.

- `-kernel_type`: the type of kernel to be used. Can be one of "graph", "sequence" and "combined".
- `-metric`: the evaluation metric to be used. Can be "RMSE" or "MAE" for regression and "ROC" or "PRC" for classification.
- `-eval_on_valid`: if included, the model will be trained on training set and evaluated on validation set; otherwise (by default), the model will be trained on the training and validation set and evaluated on the test set. Recommend to include when tuning the hyper-parameters.

V. Model configurations (hyper-parameters).

You can specify the hyper-parameters in the file `src/configs.py`. Make sure the key of your config is the same to the dataset name you specified.

In `src/configs.py`, you can add any number of configs in the form of

```
configs['(YOUR_DATASET_NAME)'] = {
    "n": (Integer. Suggest range from 1 to 12),
    "lambda": (Float. Suggest range from 0.5 to 1.0),
    "n_iters": (Integer. Suggest range from 1 to 12),
    "norm": (False/True),
    "base_k": ('subtree'/'sp')
}
```

Please replace the key `(YOUR_DATASET_NAME)` by the the name of your dataset. If the model configurations are not specified, the default setting will be used.

References

1. Cho, K. *et al.* Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1724–1734, DOI: 10.3115/v1/D14-1179 (Association for Computational Linguistics, Doha, Qatar, 2014).

2. Vaswani, A. *et al.* Attention is all you need. In *Advances in neural information processing systems*, 5998–6008 (2017).

3. Wang, S., Guo, Y., Wang, Y., Sun, H. & Huang, J. SMILES-BERT: large scale unsupervised pre-training for molecular property prediction. In *Proceedings of the 10th ACM international conference on bioinformatics, computational biology and health informatics*, 429–436 (2019).

4. Wu, Z. *et al.* MoleculeNet: a benchmark for molecular machine learning. *Chem. science* **9**, 513–530 (2018).

5. Stokes, J. M. *et al.* A deep learning approach to antibiotic discovery. *Cell* **180**, 688–702 (2020).

6. AI Cures. https://www.aicures.mit.edu/tasks. Accessed:2020-11-25.

7. Yuan, Z. & Yang, T. An end-to-end machine learning library for deep AUC optimization. https://libauc.org/. Accessed:2021-06-29.