WHEN TRANSFORMER MEETS GRAPH NEURAL NETWORKS

A PREPRINT

Yingce Xia [1], **Jinhua Zhu** [2], **Lijun Wu** [1], **Yang Fan** [2], **Shufang Xie** [1], **Yutai Hou** [3], **Tao Qin** [1]
[1] Microsoft Research Asia [2] University of Science and Technology of China [3] Harbin Institute Technology
{yinxia,lijuwu,shufxi,taoqin}@microsoft.com
{teslazhu,fyabc}@mail.ustc.edu.cn
ythou@ir.hit.edu.cn

ABSTRACT

In this paper, we present our solution to the OGB Large-Scale Challenge (OGB-LSC) at KDD Cup 2021. We mainly use three types of models: (1) Standard Transformer; (2) A two-branch Transformer, where one branch is for regression and the other is for classification. The two branches learn from each other; (3) the GIN models with virtual nodes which are provided by the organizers. The Transformers models take raw SMILES sequences as input, and the GIN models take the 2D graphs obtained through RDKit as input. We first verify our proposed network architecture and tune the hyperparameters according to the validation performance on the official validation sets, and then we merge all validation data into the training sets to finalize the models. We obtain 0.1253 MAE score on the test sets. Our code is available at `https://github.com/TransfromerMeetsGraph`.

Keywords Transformer, graph neural networks, molecule

1 Introduction

We (GNNLearner team) participated in one of the KDD Cup challenge, PCQM4M-LSC, which is to predict the DFT-calculated HOMO-LUMO energy gap of molecules based on the input molecule. In quantum chemistry, calculating the HOMO-LUMO energy gap based on 3D equilibrium structures is extremely cost. Our task is to predict this gap based on based on the 2D molecular graphs or SMILES sequence.

Considering that both Transformer and graph neural networks (GNN) achieve promising results on molecular property prediction, we choose the above two types models as backbones, and finally use ensemble to boost the performance. Specifically, we use the following models:

$\mathcal{M}_1$: The standard Transformer, which is the same as those used in natural language processing [Vaswani et al., 2017, Liu et al., 2019]. The Transformer model takes the SMILES sequences as input, which are pre-processed by canonicalized and tokenization. We follow the $\text{Roberta}_{\text{base}}$ architecture [Liu et al., 2019], which is a 12-layer network.

$\mathcal{M}_2$: A two-branch Transformer, where one branch is for regression and the other is for classification. We found that the provided energy gaps are actually discrete. Although there are more than $3M$ samples in the training set, there are only 9888 unique target values. Therefore, we can also regard the PCQM4M-LSC as a classification problem. The two branches learn from each other. At inference time, we only use the regression branch only.

$\mathcal{M}_3$: Considering the the above two Transformer models take SMILES sequences as input, we also use the GIN models [Xu et al., 2019] with virtual nodes, which take graphs as input and are complementary to sequences. We use the code provided by the organizers.

The mean absolute error (i.e., MAE) on the official validation set of the above three models are 0.1349, 0.1237 and 0.1315 respectively. We try ensemble of four $\mathcal{M}_2$ models with equal weights and obtain 0.1204 validation MAE. To verify whether the GIN models (i.e., $\mathcal{M}_3$) are complementary to Transformers models (e.g., $\mathcal{M}_2$), we further try ensemble of four $\mathcal{M}_2$ models (each with weight 0.2) and five $\mathcal{M}_3$ models (each with weight 0.04), we can obtain around 0.118 MAE. This shows the effectiveness of using Transformer and GNN together.

Finally, as mentioned by the organizers that "you can directly train your model on the validation set if you find useful", we merge the validation set to the training set and train the models for fixed epochs. We only apply this trick to $\mathcal{M}_1$ and $\mathcal{M}_2$ due to time limitation. After using ensemble of 17 models (4 standard Transformer, 8 two-branch Transformer, 5 GIN), we eventually obtain 0.1253 test MAE.

2 Our method

2.1 Data processing

For the input sequences of both standard Transformer and two-branch Transformer, we directly use the SMILES sequences. The sequences are first canonicalized[1] and then split by regular expressions[2]. For the input GNN models, we follow the official scripts to convert SMILES into graphs[3].

2.2 Models

Standard Transformer The Transformer model is the same as the Transformer used in natural language processing [Vaswani et al., 2017, Liu et al., 2019]. A Transformer layer consists of a self-attention layer and a feed-forward layer. Here we use a 12-layer model with hidden dimension 768 and feed-forward dimension 3072. For the training objective function, we use the L_1 loss, which is consistent with the evaluation metrics. The Transformer takes the tokenized SMILES sequence as input.

Graph neural networks For the graph neural network, we follow the GIN model [Xu et al., 2019] with virtual node, which is provided by `https://github.com/snap-stanford/ogb/tree/master/examples/lsc/pcqm4m`. Slightly different from the original code, we use the mean pooling instead of sum pooling (i.e., `--graph_pooling` is set as "mean"). The hidden dimension is increased to 1024.

Two-branch Transformer We also use a variant of Transformer, a two-branch Transformer. For ease of reference, let x denote the input SMILES sequence, y denote the energy gap of x, $y \in \mathbb{R}$. Let $\tilde{y}$ denote the class of x, which is obtained by the following pseudo python command: $\tilde{y} = \mathrm{int}(y//e_0)$, where $e_0 = 0.0027211385049999842$. Note that $\tilde{y}$ is an integer. Also, the label y is normalized by the mean and standard derivation on the training set, i.e., $\bar{y} = \frac{y-\mu}{\sigma}$, where $\mu = 5.690944545356371$ and $\sigma = 1.156134779510782$.

The two-branch Transformer has two independent branches with shared input embedding, where each branch is a 12-layer Transformer module. When an input SMILES x comes, the two branches will output representations h_1 and h_2 respectively, both of which are 768-dimension vectors. We apply a regression head (denoted as φ_r) to h_1 and a classification head (denoted as φ_c) to h_2. Each head is a two-layer MLP network with `tanh` activation. Denote the prediction results from the regression head and classification head as $\hat{y}_r$ (a real value) and $\hat{y}_c$ (a class) respectively. We use two types of loss functions in our method.

(1)*Prediction loss*: We convert $\hat{y}_r$ to a class label $\hat{y}_{r\to c}$ by $\hat{y}_{r\to c} = \mathrm{int}(\hat{y}_r//e_0)$, and convert $\hat{y}_c$ to a real value $\hat{y}_{c\to r}$ by $\hat{y}_{c\to r} = \frac{\hat{y}_c e_0 - \mu}{\sigma}$.

The loss function on the above two heads are

$$\ell_{\mathrm{pred}} = 10\|\bar{y} - \varphi_r(h_1)\|_1 + \log P(\tilde{y}|\varphi_c(h_2)) + 10\|\varphi_r(h_1) - \mathrm{SG}(\hat{y}_{c\to r})\|_1 + \log P(\mathrm{SG}(\hat{y}_{r\to c})|\varphi_c(h_2)), \quad (1)$$

where SG means stop gradient. In Eqn.(1), the first two terms are about fitting the regression target and classification target on the groundtruth data. The last two terms are about to learning from the other head. The weight 10 is heuristically set since regression loss is relatively small.

(2)*Dual-view similarity loss:* Considering that h_1 and h_2 are the representations of the same molecule, they should maintain some similarity. Inspired by the BYOL [Grill et al., 2020], we minimize their negative cosine similarity. Define the cosine similarity between two vectors q_1 and q_2 as $\mathtt{cos}(q_1, q_2) = \frac{q_1^\top q_2}{\|q_1\|_2\|q_2\|_2}$. The dual-view similarity loss is defined as follows:

$$\ell_{\mathrm{dual}} = -\mathtt{cos}(h_1, \mathrm{SG}(h_2)) - \mathtt{cos}(h_2, \mathrm{SG}(h_1)). \quad (2)$$

[1] `https://github.com/TransfromerMeetsGraph/GNNLearner/blob/dev/Two-branch%20Transformer/molecule/canonicalize.py`

[2] `https://github.com/TransfromerMeetsGraph/GNNLearner/blob/dev/Two-branch%20Transformer/molecule/tokenize_re.py`

[3] `https://github.com/snap-stanford/ogb/tree/master/examples/lsc/pcqm4m`

The overall loss of the two-branch Transformer is

$$\ell_{\text{all}} = \ell_{\text{pred}} + \ell_{\text{dual}}. \tag{3}$$

3 Experimental results

In this section, we first report the results on the official training and validation data. After that, we introduce the details of our submitted system.

3.1 Results on official validation set

The training details of each model are summarized in Table 1. Note that (1) batch size refers to the number of graphs in a minibatch; (2) we use Adam optimizer for all models; (3) for the two Transformer models, the learning rate gradually grows to the peak value with "warmup ratio $\times$ total steps", and then linearly decay until the maximum epochs. (4) We use V100 GPU for training.

	Standard Transformer	Two-branch Transformer	GIN
epoch	119	50	100
batch size per GPU	2048	512	256
GPUs	4	8	1
learning rate	2×10^{-4}	2×10^{-4}	10^{-3}
learning rate scheduler	linear decay	linear decay	$\times$0.25 every 30 epochs
Warmup Ratio	0.06	0.06	N/A
Number of parameters	86838026	205771785	19175430

Table 1: Training details of each model.

The results are summarized in Table 2. For the performances of single model, two-branch Transformer is the best, followed by GIN and standard Transformer. After using ensemble, the performances of standard Transformer, two-branch Transformer and GIN are 0.1241, 0.1204 and 0.1253 respectively. After combining them together, we can obtain 0.1182 validation MAE, which shows the effectiveness of all of them.

	Single model	Ensemble
Standard Transformer	0.1349	0.1241 (4 models)
Two-branch Transformer	0.1237	0.1204 (4 models)
GIN	0.1315	0.1253 (5 models)
All	—-	0.1182 (13 models)

Table 2: Validation performances of the models.

3.2 Results of the submitted system

For ease of reference, denote the official training set as $\mathcal{D}_{\text{tr}}$ and validation set as $\mathcal{D}_{\text{va}}$. Before submission, we make a last-minute bet:

1. We equally split the validation set into four groups, which are denoted as $\mathcal{D}_{\text{va},i}$, $i \in \{1, 2, 3, 4\}$. We train four two-branch Transformer models, where the training data for the i-th model is $\mathcal{D}_{\text{tr}} \cup (\mathcal{D}_{\text{va}} \backslash \mathcal{D}_{\text{va},i})$, and the validation data is $\mathcal{D}_{\text{va},i}$. The validation result of this setting is 0.118.
2. We train four standard Transformer models and four two-branch Transformer models with different random seeds on $\mathcal{D}_{\text{tr}} \cup \mathcal{D}_{\text{va}}$, and stop training after 119 (for standard) and 50 epochs (for two-branch) epochs. For GIN models, due to time limitation, we keep them the same as those in Section 3.1.

We finally ensemble the following models:

1. Four standard Transformer models trained on $\mathcal{D}_{\text{tr}} \cup \mathcal{D}_{\text{va}}$ with different seeds. The four models are then ensembled with equal weights. The ensemble model contributes the final prediction with weight $\frac{1}{12}$.

2. Four two-branch Transformer models trained on $\mathcal{D}_{\text{tr}} \cup \mathcal{D}_{\text{va}}$, each with weight $\frac{1.25}{12}$;
3. Four two-branch Transformer models trained on $\mathcal{D}_{\text{tr}} \cup \mathcal{D}_{\text{va},i}$, $i \in \{1, 2, 3, 4\}$, each with weight $\frac{1}{12}$;
4. Five GIN models trained on $\mathcal{D}_{\text{tr}}$ and then ensembled with equal weights. The ensemble model contributes to the final prediction with weight $\frac{2}{12}$.

We finally obtain 0.1253 on the test sets.

4 Conclusions and future work

In this challenge, our solution consists of three types of models, the standard Transformer, two-branch Transformer, and GIN. For the validation performance of single model, two-branch Transformer achieves the best result among the three models, which is 0.1237. The experimental results demonstrate the effectiveness of using all of them together. After that, we merge the validation set into training set, and eventually obtain 0.1253 MAE on the test sets. For future work, we will further study how to effectively combine Transformer and GNN models together.

References

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. URL `https://arxiv.org/pdf/1706.03762.pdf`.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=ryGs6iA5Km`.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733*, 2020.