Making Graph Neural Networks Worth It for Low-Data Molecular Machine Learning

Aneesh Pappu
University College London
aneesh.pappu.19@ucl.ac.uk

Brooks Paige
University College London
Alan Turing Institute
b.paige@ucl.ac.uk

Abstract

Graph neural networks have become very popular for machine learning on molecules due to the expressive power of their learnt representations. However, molecular machine learning is a classically low-data regime and it isn't clear that graph neural networks can avoid overfitting in low-resource settings. In contrast, fingerprint methods are the traditional standard for low-data environments due to their reduced number of parameters and manually engineered features. In this work, we investigate whether graph neural networks are competitive in small data settings compared to the parametrically 'cheaper' alternative of fingerprint methods. When we find that they are not, we explore pretraining and the meta-learning method MAML (and variants FO-MAML and ANIL) for improving graph neural network performance by transfer learning from related tasks. We find that MAML and FO-MAML do enable the graph neural network to outperform models based on fingerprints, providing a path to using graph neural networks even in settings with severely restricted data availability. In contrast to previous work, we find ANIL performs worse that other meta-learning approaches in this molecule setting. Our results suggest two reasons: molecular machine learning tasks may require significant task-specific adaptation, and distribution shifts in test tasks relative to train tasks may contribute to worse ANIL performance.

1 Introduction

Quantitative structure-activity relationship (QSAR) modelling consists of fitting a molecular prediction model to predict biochemically relevant function. QSAR modelling has a rich history in the field of cheminformatics: experimentally screening candidate molecules can prove expensive, and computational methods play a significant role in screening novel candidates faster and cheaper by enabling prioritization of which compounds are worth experimentally testing.

Traditional approaches to QSAR prediction have relied on 'fingerprint methods', which construct manually engineered bit vectors to represent molecular substructures. These representations are fed into off-the-shelf machine learning algorithms for downstream classification. The most popular standard in QSAR modelling is the ECFP/Morgan fingerprint (Rogers and Hahn, 2010; Morgan, 1965), which we use in this work. However, since the Merck Molecular Activity Challenge, deep learning based approaches have become mainstream; in recent years, message passing neural networks, a variant of graph neural networks, have become a popular deep learning method for learning expressive representations of molecules, which are inherently graph structured (Gilmer et al., 2017; Yang et al., 2019; Kearnes et al., 2016). Message passing neural networks learn feature representations of nodes and/or edges from an input graph, which are often combined into a representation for the entire graph. A notable example which we use in this work is the Chemprop model of Yang et al. (2019), which

Machine Learning for Molecules Workshop at NeurIPS 2020. https://ml4molecules.github.io

recently achieved state of the art results on a variety of molecular machine learning benchmarks. For details on Chemprop's message passing scheme, see Appendix A.

A main drawback of deep learning is that it requires large datasets to prevent overfitting, which is an issue in molecular machine learning because labeled biological data is expensive to collect and often sparse. Thus, manually engineered fingerprints combined with low parameter machine learning models have been traditionally well-suited for operating in low-data regimes. In particular, it has been shown that in low-data settings, fingerprint methods can outperform deep learning methods (Mayr et al., 2018; Yang et al., 2019), so the issue of dataset size is of crucial importance when building machine learning models for molecular property prediction. As such, we first investigate whether deep learning methods can compete with traditional fingerprint alternatives in low-resource settings.

2 Graph Neural Networks in Low Data Regimes

To examine whether graph neural networks outperform fingerprint alternatives in low-data settings, we examine Chemprop's performance relative to an ECFP4 fingerprint architecture on the preprocessed ChEMBL20 dataset (Bento et al., 2014; Mayr et al., 2018) at varying dataset thresholds. In both models, features computed by Chemprop and ECFP4 are inputs into a feed-forward network which outputs a property prediction.

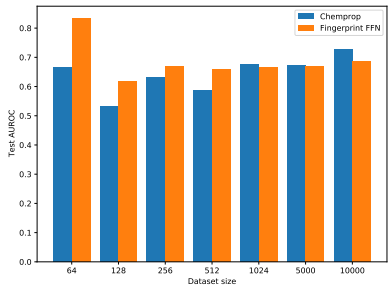

Figure 1: Performance of Chemprop versus the fingerprint method at varying dataset thresholds of the ChEMBL dataset.

Specifically, we compute the average AUROC of the fingerprint architecture and Chemprop trained jointly on all tasks with fewer datapoints than each threshold. These results are shown in Figure 1. We find that Chemprop is indeed outperformed by the fingerprint baseline at lower data thresholds, and as dataset size increases past 1024 datapoints, Chemprop becomes competitive and outperforms the fingerprint baseline. As the fingerprint method outperforms the graph neural network in smaller datasets, we shift our investigation into whether we can boost the performance of Chemprop in smaller data settings.

3 Improving Low Data Performance of Graph Neural Networks

Pretraining One strategy for mitigating deep learning's large data requirement is to leverage multitask pretraining, which combines data from multiple tasks to learn robust, general-purpose representations for future tasks. Perhaps the most famous example of pretraining is the practice of pretraining AlexNet (Krizhevsky et al., 2012) on the ImageNet classification dataset (Deng et al., 2009). Multitask learning has been explored in drug discovery and has seen success, with the caveat that models can exhibit positive or negative transfer on new tasks (Ramsundar et al., 2015). We use the Chemprop architecture proposed in Yang et al. (2019) as our pretraining architecture (full hyperparameter details are in Appendix D).

Meta-Learning Meanwhile, meta-learning methods (Schmidhuber, 1987; Hinton and Plaut, 1987; Vinyals et al., 2016; Lake et al., 2015; Altae-Tran et al., 2017) have gained significant traction within the few-shot learning community, with the aim of leveraging prior tasks to 'learn to learn' so that good performance on future tasks can be obtained with few datapoints. In particular, gradient-based meta-learning algorithms (Ravi and Larochelle, 2017; Finn et al., 2017) have gained popularity, with perhaps the most recent significant advance coming from the Model Agnostic Meta-Learning (MAML) algorithm and its close variant First-Order MAML (FO-MAML) (Finn et al., 2017).

MAML and FO-MAML MAML and FO-MAML frame meta-learning as optimizing for a network initialisation that is capable of learning quickly on future tasks. MAML consists of a meta-training phase, where train tasks T^{tr} are used to learn a meta-initialisation via an 'outer loop' and 'inner loop', and a meta-testing phase, where test tasks T^{test} are used to evaluate how well the model adapts to new tasks. The inner loop calculates task specific updates to the meta-initialisation, and the outer loop calculates meta-gradients with respect to the meta-initialisation via the updated model parameters in order to update the meta-initialisation to an initialisation capable of learning quickly (further

detail in Appendix B). As the outer loop update is quite expensive due to computing second-order gradients, Finn et al. (2017) propose FO-MAML, which omits second order gradient terms from the meta-update.

ANIL: Feature Reuse vs. Rapid Learning Raghu et al. (2019) conclude that MAML's efficacy arises from learning reusable features as opposed to features capable of rapid learning. They found that across supervised and reinforcement learning tasks, MAML-trained initialisations adapt very little during test task-specific learning, as evidenced by high CCA similarity between pre and post adaptation layers. As a result, Raghu et al. (2019) propose the ANIL algorithm, which removes the inner loop for all layers but the head layer, and observe near identical performance to MAML.

Dataset and Experiment We pretrain Chemprop and implement MAML, FO-MAML, and ANIL on Chemprop to see whether these approaches boost Chemprop's performance relative to the fingerprint method. We filter the ChEMBL20 (Bento et al., 2014; Mayr et al., 2018) dataset by tasks that have between 128 and 1024 datapoints. This results in a new dataset consisting of 645 binary classification tasks across 5 distinct task types obtained from the ChEMBL database: ADME (A), Toxicity (T), Binding (B), Functional (F), and Unassigned (U). Following Nguyen et al. (2020), we split the 645 tasks into three task splits, $T^{tr}, T^{val}, T^{test}$. T^{test} is composed of 10 randomly assigned B and F tasks and all of the A, T, and U tasks. T^{val} is randomly assigned 10 B and F tasks, and the remaining B and F tasks are assigned to T^{tr}. This allows us to assess performance of our methods on both in-distribution tasks (by performance on the B and F tasks in T^{test}) and out-of-distribution tasks (by performance on the A, T, and U tasks in T^{test}). We use a scaffold split for all *within* task splits. A summary of the task split is shown in Table 4 and further dataset details are included in Appendix C.

Evaluation The performances of the fingerprint method, pretrained Chemprop, and all meta-learning methods are shown in Table 1 and Table 2 for the in-distribution and out-of-distribution tasks, respectively. We summarize the average rank of each method in Table 3 and use Wilcoxon signed-rank testing to assess significance of rank differences on all pairwise method performances, with p-values shown in Table 5. As shown by average ranks and Wilcoxon testing, we find that pretraining *doesn't* significantly outperform the fingerprint baseline. However, we find that MAML and FO-MAML *significantly outperform* both the pretraining and fingerprint methods, showing that meta-learning is able to boost the performance of the graph neural network in this low-data setting.

Interestingly, we find that ANIL performs consistently worse than MAML and is not statistically significantly different from the fingerprint or pretrained models (Table 5). This is in contrast to the findings of Raghu et al. (2019) that ANIL suffers no loss in performance compared to MAML. This discrepancy motivates our next investigation into why ANIL performs poorly in this molecule setting.

Table 1: In distribution mean AUPRC with standard deviations across five folds. Best performing result is bold text and second best is regular text. 'Frac. Pos.' denotes the fraction of positives in the dataset (expected AUPRC of a random classifier). '# Obs' is number of datapoints in the task.

ChEMBL ID	ECFP	Pretraining	MAML	FO-MAML	ANIL	Frac. Pos.	# Obs
1738202	0.953 ± 0.086	0.909 ± 0.050	**0.997 ± 0.006**	0.991 ± 0.013	0.942 ± 0.032	0.965	144
3215176	0.542 ± 0.270	0.362 ± 0.097	0.482 ± 0.342	**0.576 ± 0.271**	0.431 ± 0.360	0.083	157
1963934	**0.957 ± 0.022**	0.865 ± 0.091	0.863 ± 0.113	0.883 ± 0.057	0.928 ± 0.026	0.964	165
1794358	0.264 ± 0.218	0.415 ± 0.331	**0.577 ± 0.295**	0.517 ± 0.321	0.337 ± 0.336	0.059	222
2114797	**0.868 ± 0.107**	0.805 ± 0.073	0.796 ± 0.036	0.827 ± 0.074	0.569 ± 0.041	0.576	224
3215116	0.167 ± 0.070	0.152 ± 0.078	0.210 ± 0.117	0.292 ± 0.218	**0.484 ± 0.179**	0.161	248
1794355	0.935 ± 0.068	**0.954 ± 0.027**	0.932 ± 0.029	0.895 ± 0.045	0.932 ± 0.035	0.947	304
1614202	0.897 ± 0.086	0.928 ± 0.084	0.900 ± 0.076	0.874 ± 0.115	**0.957 ± 0.047**	0.927	314
1794567	0.920 ± 0.036	**0.939 ± 0.051**	0.923 ± 0.067	0.923± 0.068	0.924 ± 0.078	0.904	385
1614359	**0.728 ± 0.147**	0.621 ± 0.117	0.702 ± 0.167	0.681 ± 0.168	0.628 ± 0.185	0.497	390
1738131	0.476 ± 0.085	0.446 ± 0.121	0.556 ± 0.146	**0.631 ± 0.087**	0.444 ± 0.171	0.314	468
1614170	0.364 ± 0.102	0.360 ± 0.118	**0.456 ± 0.088**	0.395 ± 0.155	0.310 ± 0.093	0.311	546
1963705	0.562 ± 0.109	0.775 ± 0.045	0.807 ± 0.075	0.765 ± 0.078	**0.862 ± 0.046**	0.441	692
1909212	0.049 ± 0.021	0.150 ± 0.155	**0.185 ± 0.121**	0.078 ± 0.034	0.050 ± 0.017	0.017	824
1909209	0.264 ± 0.136	0.155 ± 0.088	0.527 ± 0.192	**0.544 ± 0.276**	0.275 ± 0.164	0.077	835
1909085	0.247 ± 0.155	0.408 ± 0.176	**0.811 ± 0.169**	0.668 ± 0.267	0.254 ± 0.118	0.079	835
1909192	0.013 ± 0.004	**0.278 ± 0.370**	0.231 ± 0.385	0.045 ± 0.019	0.026 ± 0.037	0.004	838
1909092	0.066 ± 0.073	0.084 ± 0.099	0.580 ± 0.366	**0.797 ± 0.284**	0.059 ± 0.077	0.029	838
1909211	0.497 ± 0.134	0.780 ± 0.072	**0.802 ± 0.108**	0.771 ± 0.133	0.423 ± 0.134	0.112	839
1963741	0.461 ± 0.053	0.554 ± 0.141	0.682 ± 0.075	**0.718 ± 0.080**	0.644 ± 0.060	0.330	919

Table 2: Out of distribution mean AUPRC with standard deviations across five folds. Best performing result is bold text and second best is regular text. 'Frac. Pos.' denotes the fraction of positives in the dataset (expected AUPRC of a random classifier). '# Obs' is number of datapoints in the task.

ChEMBL ID	ECFP	Pretraining	MAML	FO-MAML	ANIL	Frac. Pos.	# Obs
2098499	0.516±0.274	0.500±0.159	0.508±0.190	**0.593±0.188**	0.483±0.231	0.255	137
1738021	0.899±0.079	0.958±0.033	**0.973±0.025**	0.888±0.088	0.885±0.118	0.891	138
1738019	0.731±0.153	0.682±0.136	0.791±0.088	**0.903±0.127**	0.793±0.158	0.752	165
918058	0.282±0.361	0.205±0.196	0.429±0.393	**0.626±0.458**	0.036±0.016	0.067	225
2095143	0.440±0.309	0.074±0.062	0.418±0.309	**0.539±0.282**	0.418±0.290	0.062	273
2028077	0.040±0.011	0.057±0.023	0.300±0.254	0.494±0.353	**0.508±0.324**	0.038	289

Table 3: Average rank of each method within the subsets of in-distribution tasks, out-of-distribution tasks, and all tasks. Bold is best (lowest) average rank and regular text is second best average rank.

Task Subset	ECFP	Pretraining	MAML	FO-MAML	ANIL
In-distribution	3.55	3.25	**2.20**	2.50	3.50
Out-of-distribution	3.17	4.00	2.67	**1.67**	3.50
All	2.70	2.10	**1.43**	1.79	2.37

4 Feature Reuse vs. Rapid Learning in Molecular Machine Learning

CCA Similarity To investigate ANIL's poor performance, we study task adaptation versus feature reuse via measuring layer representation similarity pre and post adaptation as in Raghu et al. (2019). Specifically, we use our MAML trained initialisation to adapt the network on each test task across 5 seeds, and calculate the CCA similarity between each layer of the graph neural network pre and post adaptation. As the message passing network maintains a variable number of hidden representations based on the number of atoms in a molecule, we calculate similarity based on the final molecule level representation of the graph neural network, which is size invariant to the number of atoms in the molecule. We also compute CCA similarity coefficients for both feed-forward layers.

The similarity results are shown in Figure 2. Compared to the CCA similarity experiment in Raghu et al. (2019), the average similarity for each layer is much lower than what we would expect if inner loop adaptation wasn't necessary. Raghu et al. (2019) observes that for all layers except the head, CCA similarity is nearly 1. In contrast, we see that the graph neural network and first feed-forward layer have changed significantly after inner loop adaptation. These findings suggest that ANIL performs poorly in this molecule setting because inner loop adaptation is necessary for the body layers, and that distribution shift relative to the meta-train tasks adversely affects ANIL's effectiveness, as there is decreased layer similarity when solely examining the out of distribution tasks.

5 Conclusion and Future Work

In this work we investigated whether graph neural networks are useful compared to fingerprint methods in low-data settings. We found that while graph neural networks underperform relative to fingerprint methods, MAML and FO-MAML significantly improve performance, outperforming both the fingerprint and pretraining methods. This shows meta-learning enables use of graph neural networks in low-data settings over fingerprint methods. We also find that ANIL, contrary to prior belief, does not match performance of MAML. Our results suggest that inner loop adaptation is important in this molecule setting and that ANIL's performance is affected by distribution shift in test tasks. Future work includes investigating whether ANIL breaks in the supervised and reinforcement learning settings investigated in Raghu et al. (2019) when domain shift is engineered into the meta-task splits.

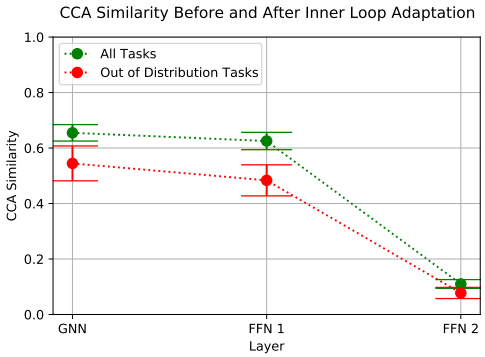

Figure 2: CCA similarity pre and post inner loop adaptation across all tasks and solely the out of distribution tasks.

Acknowledgments and Disclosure of Funding

We would like to acknowledge Cuong Nguyen and the GSK Artificial Intelligence team, Kyle Swanson, Bharath Ramsundar, Yihong Chen, Luca Franceschi, and Pasquale Minervini for helpful comments and feedback. This work was supported by The Alan Turing Institute under the EPSRC grant EP/N510129/1. We would also like to thank the Marshall Aid Commemoration Commission for providing a Marshall Scholarship to fund AP.

References

Altae-Tran, H., Ramsundar, B., Pappu, A. S., and Pande, V. (2017). Low Data Drug Discovery with One-Shot Learning. *ACS Central Science*, 3(4):283–293.

Arnold, S., Mahajan, P., Datta, D., and Bunner, I. (2019). learn2learn.

Bento, A. P., Gaulton, A., Hersey, A., Bellis, L. J., Chambers, J., Davies, M., Krüger, F. A., Light, Y., Mak, L., McGlinchey, S., et al. (2014). The chembl bioactivity database: an update. *Nucleic acids research*, 42(D1):D1083–D1090.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *34th International Conference on Machine Learning, ICML 2017*, volume 3, pages 1856–1868. International Machine Learning Society (IMLS).

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *34th International Conference on Machine Learning, ICML 2017*, volume 3, pages 2053–2070.

Hinton, G. E. and Plaut, D. C. (1987). Using fast weights to deblur old memories. In *Proceedings of the ninth annual conference of the Cognitive Science Society*, pages 177–186.

Kearnes, S., McCloskey, K., Berndl, M., Pande, V., and Riley, P. (2016). Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338.

Mayr, A., Klambauer, G., Unterthiner, T., Steijaert, M., Wegner, J. K., Ceulemans, H., Clevert, D. A., and Hochreiter, S. (2018). Large-scale comparison of machine learning methods for drug target prediction on ChEMBL. *Chemical Science*, 9(24):5441–5451.

Morgan, H. L. (1965). The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service. *Journal of Chemical Documentation*, 5(2):107–113.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *ICML*.

Nguyen, C. Q., Kreatsoulas, C., and Branson, K. M. (2020). Meta-Learning Initializations for Low-Resource Drug Discovery.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

Raghu, A., Raghu, M., Bengio, S., and Vinyals, O. (2019). Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML.

Ramsundar, B., Kearnes, S., Riley, P., Webster, D., Konerding, D., and Pande, V. (2015). Massively multitask networks for drug discovery. *arXiv preprint arXiv:1502.02072*.

Ravi, S. and Larochelle, H. (2017). Optimization as a model for few-shot learning. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*.

Rogers, D. and Hahn, M. (2010). Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling*, 50(5):742–754.

Schmidhuber, J. (1987). Evolutionary principles in self-referential learning. *On learning how to learn: The meta-meta-... hook.) Diploma thesis, Institut f. Informatik, Tech. Univ. Munich*, 1(2).

Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., and Wierstra, D. (2016). Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3637–3645.

Yang, K., Swanson, K., Jin, W., Coley, C., Eiden, P., Gao, H., Guzman-Perez, A., Hopper, T., Kelley, B., Mathea, M., Palmer, A., Settels, V., Jaakkola, T., Jensen, K., and Barzilay, R. (2019). Analyzing Learned Molecular Representations for Property Prediction. *Journal of Chemical Information and Modeling*, 59(8):3370–3388.

Appendix

A Chemprop Message Passing Details

Concretely, for an input graph G with nodes v, w, message update function M_t and hidden state update function U_t, each iteration of message passing computes updates as

$$m_{vw}^{t+1} = \sum_{k \in \{N(v) \setminus w\}} h_{vw}^{t}, \qquad h_{vw}^{t+1} = \tau(h_{vw}^{0} + W_m m_{vw}^{t+1}) \tag{1}$$

where τ is the ReLU nonlinearity (Nair and Hinton, 2010), and W_m is a learnable matrix. The initial hidden state is computed as $h_{vw}^{0} = \tau(W_i(x_v, e_{vw}))$ where W_i is a learnable matrix and x_v and e_{vw} correspond to input node and edge features. After the prespecified iterations of message passing have concluded, Chemprop computes the atom-level representations as

$$m_v = \sum_{k \in N(v)} h_{kv}, \qquad h_v = \tau(W_a(x_v, m_v)) \tag{2}$$

where W_a is a learnable matrix. Finally, the graph-level representation is calculated via a sum-pool operation, $h_G = \sum_{v \in G} h_v$. Afterwards, the readout phase computes the prediction $\hat{y}$ by applying a feed-forward neural network to the graph representation.

B MAML Algorithm Details

During meta-training, the model is initialised with parameters θ and a batch of tasks $\{T_i\}$ is sampled. For each task, a *support* and *query* set are sampled. The model is updated with respect to the loss on the support set of the parameters θ using the standard SGD rule:

$$\theta_i^{'} = \theta - \alpha \nabla_\theta L_{T_i}(\theta)$$

where $\theta_i^{'}$ are the new parameters for task T_i computed using the gradient of the loss on the support set of task T_i and α is the inner loop learning rate. This is repeated for each meta-train task, producing a set of parameters $\{\theta_i^{'}\}$ for each meta-train task T_i. These updated parameters $\theta_i^{'}$ are then used to calculate the loss on the query set for each task which represents a task-specific validation loss. This loss acts as the meta-loss, which is used to calculate the gradient with respect to the *original* parameters θ to update the meta-initialisation θ:

$$\theta = \theta - \beta \nabla_\theta \sum_{T_i} L_{T_i}(\theta_i^{'})$$

where β is the outer loop learning rate.

C Meta Task Splits and Dataset Details

Table 4 shows the number of tasks in each task type allocated to each meta-task split.

Table 4: Number of tasks in each meta-task split by task type.

	A	*T*	*U*	*B*	*F*
T^{train}	0	0	0	128	471
T^{val}	0	0	0	10	10
T^{test}	2	2	2	10	10

In the meta-train and meta-validation splits, we use a within-task split ratio of 80% train and 20% validation. In the meta-test split we use a split ratio of 80% train, 10% validation, and 10% test. This is because during meta-training and meta-validation, we only need a support set for task-specific adaptation and a query set for calculating meta-loss and meta-gradients, but in the meta-test phase we require a train and validation set for training and early stopping, and a test set for calculating final test performance. For the pretraining method, we combine T^{tr} and T^{val} which is then split into D^{tr} for training and D^{val} for early stopping and hyperparameter tuning. We train the fingerprint architecture from scratch for each of the test tasks in T^{test}. Each method is evaluated over 5 seeds on each task in T^{test}.

D Hyperparameters

We use the ADAM optimizer (Kingma and Ba, 2014) for all optimizations. We use Learn2Learn (Arnold et al., 2019) and PyTorch (Paszke et al., 2017) for our meta-learning implementations.

D.1 Fingerprint Architecture

We use ECFP4 fingerprints for our input fingerprints. The final fingerprint hyperparameters after grid search are a dropout rate of 0.2 and a hidden layer size of 400. We use a batch size of 32 and a learning rate of 10^{-4} for all fingerprint experiments.

D.2 Pretraining Architecture

The final pretraining hyperparameters after grid search are a dropout rate of 0.2 and 2 message passing steps. We fix the number of feed-forward layers used in the readout phase of the graph neural network to be 2 so that it matches the fingerprint baseline architecture. We use a batch size of 32 and a learning rate of 10^{-4} for all experiments.

D.3 Meta-Learning Models

The meta-learning models use the same hyperparameters as the pretraining architecture as the meta-learning methods are implemented on top of Chemprop. The primary meta-learning hyperparameters we tune via grid search are the outer and inner loop learning rates. In this work we use an outer loop learning rate of 10^{-3} and an inner loop learning rate of 0.05. At meta-test time, we use a learning rate of 10^{-4}. We use a meta batch size of 32 (i.e., each outer loop update happens using meta-gradients on 32 tasks), and an inner loop batch size of 32 (i.e., 32 datapoints per task).

E Wilcoxon Signed Rank Test

Table 5 shows the p-values computed for each pairwise grouping of methods. In particular, each method is represented as a sample of 130 datapoints where each datapoint is the AUPRC on one seed on one of the 26 test tasks. We represent each method by this vector $\in \mathbb{R}^{130}$ for the Wilcoxon signed-rank test. We use a significance threshold of 0.05.

Table 5: P-values of Wilcoxon signed-rank test performed pairwise across all methods. Signed-rank test computed with respect to each method's performance across all 26 test tasks and the 5 seeds on each task. Threshold for significance is 0.05, and significant p-values are bold.

	ECFP	Pretraining	MAML	FO-MAML	ANIL
ECFP	*	0.73	**0.002**	**0.001**	0.83
Pretraining	0.73	*	**0.0003**	**0.004**	0.99
MAML	**0.002**	**0.0003**	*	0.47	**0.01**
FO-MAML	**0.001**	**0.004**	0.47	*	**0.004**
ANIL	0.83	0.99	**0.01**	**0.004**	*