NNTI Project team 100

Samuele Serri – 7069839 – sase00007@stud.uni-saarland.de
Bharath Vasishta Iriventi – 7058181 – bhir00001@stud.uni-saarland.de
Omid Nezhadbahadori – 7062316 – omne00001@stud.uni-saarland.de

Abstract

In this report, we have carried out 3 tasks for implementing and evaluating different regression Chemical Language Models for the Lipophilicity dataset, part of MoleculeNet benchmark, using neural networks in which we have fine-tuned a pre-trained model, add external data to improve the performance of the base model respectively, and implemented and compared different strategies in data selection and some parametric-efficient fine-tuning approaches. Lastly, we have made a conclusion based on the observations to support the outcome of the tasks.

Introduction

Lipophilicity is a fundamental property in drug discovery, influencing the absorption, distribution, metabolism, and excretion (ADME) of compounds. The Lipophilicity dataset is part of the **MoleculeNet** benchmark (Wu et al., 2018), which provides a datasets for evaluating machine learning models in molecular property prediction. Traditional approaches was based on handmade molecular descriptors and physicochemical properties, but recent advances in deep learning have introduced chemical language models capable of learning molecular representations directly from data.

The dataset contains **4200 samples**, with each sample having two features: **"SMILES"** and **"label"**. The "SMILES" (Simplified Molecular Input Line Entry System) represents a molecular structure. The "label" represents the experimental $\log P$ (octanol-water partition coefficient), a numerical value that quantifies the compound's lipophilicity (Wu et al., 2018). The goal of this task is to build a regression model that accurately predicts $\log P$ based on the molecular structure encoded in SMILES.

In this work, we focus on implementing and improving a regression-based Chemical Language Model for lipophilicity prediction. The tasks are:

1. **Fine-tuning a pre-trained model** on the Lipophilicity dataset to establish a baseline. Then, **add unsupervised fine-tuning** on the base model to enhance the performance.

2. **Enhancing model performance by adding external data** by calculating **Influence Score**, and tackling computational challenges using **LiSSA** approximation.

3. **Implementing and comparing different strategies** for data selection and parametric-efficient fine-tuning, including **BitFit**, **LoRA**, and **iA3**.

By evaluating these techniques, we aim to identify effective strategies for improving lipophilicity prediction while maintaining computational and memory efficiency. The results of this study contribute to the broader application of neural networks in cheminformatics and molecular property prediction.

1 Task 1

In the first task, we have fine-tuned a pre-trained model to make a baseline. Then, we have added a unsupervised masked linear model to enhance the performance to the model in order to compare the output.

1.1 Loading and preprocessed the dataset

To load the dataset, we used the `datasets` library from Hugging Face.

Then, we split the dataset into three sets of **training set**, **validation set**, and **test set**. The **training set** and **validation set** are used to train the model and evaluate the training process, and **test set** to evaluate the whole trained model using unseen data. We have used 75% of the data (3050 samples) in training process and 25% of the data (1050 samples) for test process. 90% of the training data

(2745 samples) in training process is used for training and the rest 10% (305 samples) for validating.

Finally, we have loaded the data with **batch size** of 16.

1.2 Loading a pre-trained transformer model

We load the pre-trained model using the `AutoModel` class from the Hugging Face `transformers` library. Since the models are pre-trained for classification or masked language modeling, we have added a regression head to prepare the transformer model for regression task.

1.3 Training and Evaluating the pre-trained model

We fine-tune the pre-trained model with training it with the specific dataset to prepare it for our task. To train the model, we have trained the model on training data and evaluated the training process using validation set. Before feeding the data into the model, we first tokenized the SMILES representations using a pre-trained tokenizer. Using 5 epochs and early stopping with patience of 3 and learning rate of 0.0001, we have achieved the final test loss of **1.5213** as the baseline for the fine-tuned pre-trained model.

1.4 Adding Unsupervised Finetuning

In this part, we have performed unsupervised fine-tuning on the training dataset to enhance the model's adaption of understanding of the data distribution. Using Masked Language Modeling MLM objective, the model learns to predict randomly masked tokens within the input sequence. We have used learning rate of 0.0001, probability of 0.15 to mask 15% of tokens, and 3 epochs. The final test loss was **0.1897**.

1.5 Fine-tuning for the comparison

We have trained the unsupervised fine-tuned model to compare the trained model with the one that we have fine-tuned in previously. The final test loss was **0.4137**

The comparison of the results is shown in Figure 1.

2 Task 2

In this task we compute influence functions in order to select a subset of the external dateset and augment the original training dataset. Influence functions are computed as $-\nabla_\omega L(z_{test}, \omega^*)^T H_{\omega^*}^{-1} \nabla L(z, \omega^*)$(Pang Wei Koh, 2020). Our Neural Network has $\approx 44 \times 10^6$ parameters, hence, inverting the Hessian, which requires $O(p^3)$ operations, is infeasible. We approximate $H_{\omega^*}^{-1} \nabla L(z, \omega^*)$ with the algorithm illustrated in(Naman Agarwal, 2017).
In particular we set $\tilde{H}_0^{-1} \nabla L = \nabla L$ and $\tilde{H}_t^{-1} \nabla L = \nabla L + \tilde{H}_{t-1}^{-1} \nabla L - \nabla^2 L \tilde{H}_{t-1}^{-1} \nabla L$ [1], where $\nabla^2 L$ is an unbiased estimator of the hessian. To compute this last quantity we use torch automatic differentiation. We find that, based on the weights w^* from Task 1 fine-tuned model, we obtain the influence scores distribution showed in Figure 2. We decide to include in the dataset all the training points corresponding to an influence score $\geq [5, 6.5, 7.5]$ in absolute value which are 77%, 50% and 33% of the external dataset respectively. Fine-tuning the model with the new datasets we observe the behavior showed in Figure 3 for the first case, Figure 4 for the second case and Figure 5 for the third case. The RMSE comparison is showed in Figure 6.

3 Task 3

We explore an alternate method from Task 2 for data selection and investigate three Parameter Efficient Fine-tuning (PEFT) methods to adapt a pre-trained model for a specific task.

3.1 Data Selection Method

We efficiently select the most valuable 100 samples from a pool of 300 by quantifying **prediction uncertainty through Monte Carlo dropout.** For each molecule in our dataset, we run the model 10 times with dropout enabled during inference, creating a distribution of predicted values for each sample. We then calculate the standard deviation of these predictions as a direct measure of uncertainty, a higher standard deviation indicates the model is less confident about that particular sample. By ranking all 300 samples according to their uncertainty scores and selecting the top 100 with the highest uncertainty values, we identify precisely those examples where the model struggles most. These high-uncertainty samples represent the decision boundary regions where additional training would most effectively improve our model's performance, making them ideal candidates for targeted fine-tuning.(3)

[1] In our code all vectors are further normalized

3.2 Parameter-Efficient Fine-Tuning (PEFT)

3.2.1 BitFit (Bias Terms Fine-Tuning)

BitFit involves fine-tuning only the bias terms of the model's layers, keeping the majority of parameters frozen. This approach reduces computational resources and mitigates overfitting, making it efficient for tasks with limited data. (Elad Ben-Zaken, 2022)

3.2.2 LoRA (Low-Rank Adaptation)

LoRA introduces trainable low-rank matrices into each layer of the neural network, allowing adaptation to specific tasks without modifying the original weights extensively. This method maintains model performance while significantly reducing the number of trainable parameters, enhancing efficiency. (Edward Hu, 2021)

3.2.3 iA3(Infused Adapter by Inhibiting and Amplifying Inner Activations")

iA3 focuses on adapting transformer activations by introducing task-specific parameters that rescale the keys, values, and feed-forward activations. This allows mixed-task batches without modifying model weights, enabling efficient adaptation to new tasks while preserving computational efficiency.(Haokun Liu, 2022)

3.3 Results

3.3.1 Training and Performance

Each model was trained for 5 Epochs with a learning rate of 0.001. Time and computational resources were taken into factor for the Hyperparameters. The performance of each fine-tuned model was evaluated using mean squared error (MSE) and R-squared (R^2) metrics on a held-out test set.

Method	Trainable Parameters
Standard FT	44,375,809
LoRA	1,474,560
BitFit	74,497
iA3	27,648

Table 1: Trainable Parameters for Different Methods

3.3.2 Analysis

BitFit outperformed IA^3 and LoRA due to its efficiency in updating only the bias terms, allowing for quicker convergence and better performance with fewer parameters. BitFit achieved a 13.5 % improvement in RMSE, likely because its smaller optimization space allowed it to adapt faster within the 5 epochs. LoRA, which updates low-rank matrices, also showed good performance with a 5.9% RMSE improvement, but BitFit still had a more significant impact. In comparison, IA^3's multiplicative activation rescaling required more training time to effectively adapt, which may have contributed to its more modest 6.5% RMSE improvement. The limited 5-epoch training period likely led to underfitting for IA^3, whereas BitFit's simpler adjustments helped it achieve better results in this short time frame. Look at Figure 7.

References

Uncertainty Estimation in Machine Learning with Monte Carlo Dropout. [link].

Phillip Wallis Edward Hu, Yelong Shen. 2021. Lora: Low-rank adaptation of large language m odels.

Yoav Goldberg Elad Ben-Zaken, Shauli Ravfogel1. 2022. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models.

Jay Mohta Haokun Liu, Derek Tam. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning.

Elad Hazan Naman Agarwal, Brian Bullins. 2017. Second-order stochastic optimization for machine learning in linear time.

Percy Liang Pang Wei Koh. 2020. Understanding black-box predictions via influence functions.

Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. 2018. Moleculenet: A benchmark for molecular machine learning. *Chemical Science*, 9(2):513–530.

Plots

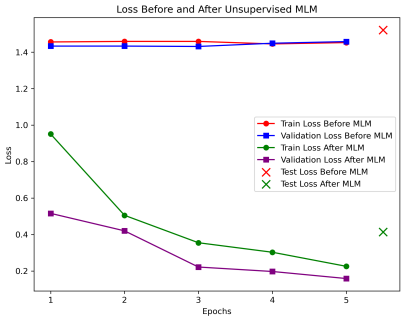

Figure 1: Comparison of training, validation, and test loss before and after unsupervised MLM.

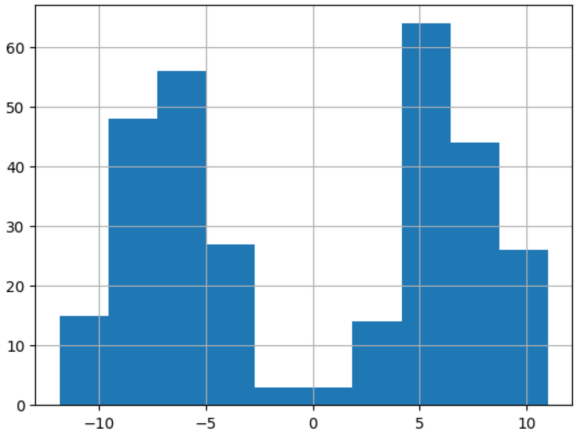

Figure 2: Influence scores distribution

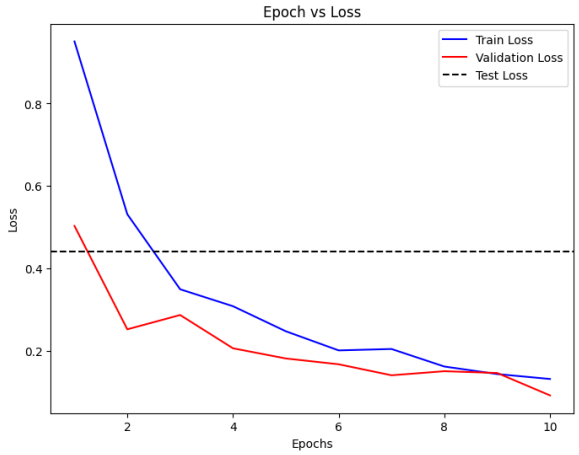

Figure 3: Train and validation loss (MSE) with ~77% of the external dataset.
lr = 0.0001

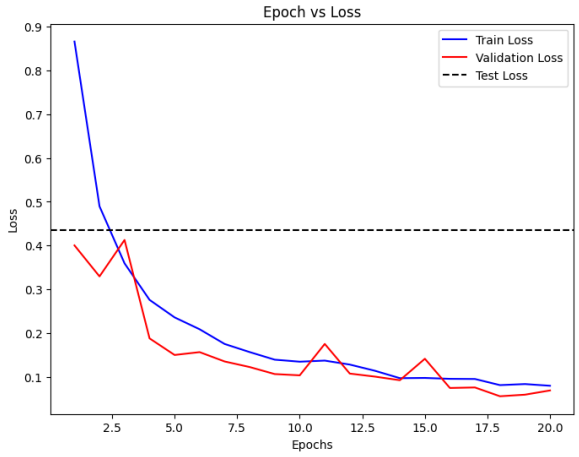

Figure 4: Train and validation loss (MSE) with ~50% of the external dataset.
lr = 0.0001

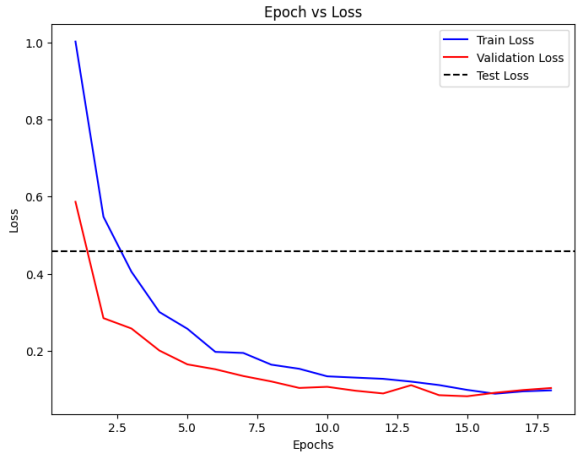

Figure 5: Train and validation loss (MSE) with ~33% of the external dataset.
lr = 0.0001

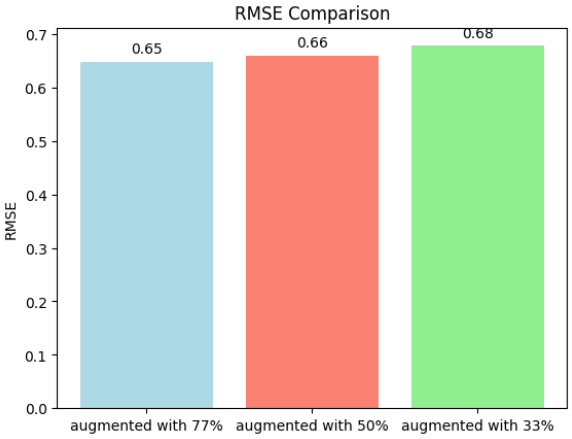

Figure 6: RMSE Comparison Task 2

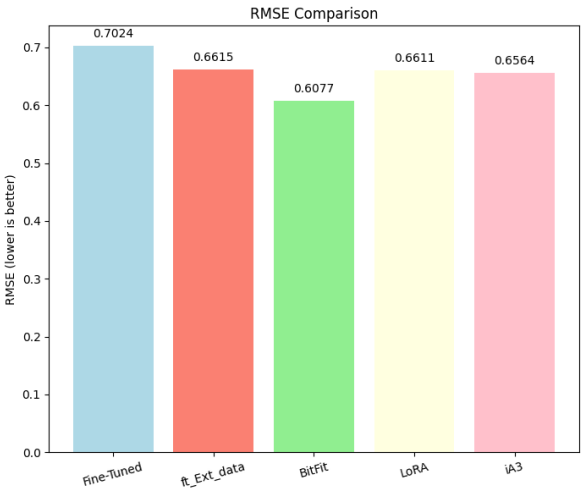

Figure 7: RMSE Comparison Task 3