NNTI Project WS 2024/2025

Michael Gallinger 7019676 (miga00001)
Nischal Narendra Giriyan 7048036 (nigi00005)
Shreyansh Chakraborty 7025878 (shch00002)@stud.uni-saarland.de

1 Introduction

In this project, our goal is to predict the lipophilicity value of a molecule using a neural network, with a focus on a transformer model. To start, we will explore a straightforward approach by using a pretrained transformer model with minimal fine-tuning. This allows us to establish a baseline performance before diving deeper into model optimization. Once we have the initial results, we will investigate several advanced techniques to improve the performance of the model for this specific task. Enhancing model accuracy involves not only fine-tuning the model itself but also improving the quality and relevance of the data used during training. As part of our approach, we will experiment with the following strategies:
-MLM-Pretraining
-Influence Function-based Data Selection (Task 2)
-Data selection strategies
-BitFit, LoRA and iA^3 PEFTs

2 General Model with Regression Head (Task 1)

To predict the lipophilicity value of a given molecular representation, we started with the pretrained model "ibm/MoLFormer-XL-both-10pct". This transformer model is already trained on chemical language, represented as SMILES strings. Our goal was to fine-tune this base model for our specific task. To achieve this, we added a regression head to the base model and fine-tuned it using the Lipophilicity dataset, which is part of the MoleculeNet benchmark.

2.1 Model and Data Preparation

The dataset was split into training, validation, and test sets. The validation set was used for regularization via early stopping. Additionally, we defined a PyTorch Dataset class to tokenize and preprocess the data for training. Each dataset was then loaded into a DataLoader for efficient processing.

Regarding the model, we implemented a PyTorch model class that takes the pretrained model as an argument and appends a single linear layer (the "Regression Head") after the last hidden layer.

2.2 Model Training

The training procedure follows a standard approach with some additional details that are important to mention:

We used AdamW as the optimizer and MSELoss as the criterion. AdamW is the standard optimizer for transformer models, while MSELoss is the typical loss function for regression tasks.

To support evaluation and comparison between models, we additionally stored the average training and validation losses for each epoch.

2.2.1 Hyperparameters

-**Batch size:** 32, chosen to balance accuracy and computational efficiency.
-**Learning rate:** 2×10^{-5}, selected based on empirical experimentation to achieve the best accuracy while keeping the number of training epochs reasonable (roughly around 10-25 epochs).
-**Early stopping:** A patience of 5 epochs was applied. After each epoch, we evaluated the validation loss, and if no improvement was observed for five consecutive epochs, training was interrupted. The model with the lowest validation loss was selected as the best generalizing one with a reasonable low training loss 1.
-**Weight decay:** 0.5, also determined through empirical experimentation to introduce more regularization and reduce the risk of overfitting.

2.3 Model Evaluation

For evaluation, we computed the Mean Squared Error (MSE), Mean Absolute Error (MAE), and the R^2-score to provide a balanced view of the model performance. Since MSE effectively penalizes large errors, MAE is robust and interpretable and the R^2-score quantifies how much of the variance in the data is explained. Additionally, we created a plot comparing true versus predicted values, including an optimal prediction line with a margin, based on the median absolute error. Furthermore, we plotted the average training and validation losses over the training epochs 2.

3 MLM-Pretraining

To enhance model performance, we introduced an additional step: unsupervised MLM (Masked Language Modeling) pretraining before fine-tuning with the regression head. In this pretraining phase, we used only the molecule representations from our dataset (without labels), randomly masking parts of the SMILES strings and training the model to predict the missing sections. This step aimed to improve the model's understanding of the dataset, resulting in a overall better performance.

3.1 MLM-Pretraining Process

We first prepared the dataset by removing labels, splitting it into training, validation, and test sets, applying random masking, tokenizing the sequences, and loading them into a DataLoader. We then reloaded the base model and trained it using CrossEntropyLoss as the loss function. Since MLM is a multi-class classification task (predicting missing parts of the SMILES strings).

3.2 Fine-Tuning and Comparison

Once the MLM-pretraining was completed, we followed the same procedure as before: adding a regression head, training on the same dataset, and using the same number of epochs. Finally, we evaluated both fine-tuned models with regression heads for comparison.

3.3 Results

The MLM-pretrained model outperformed the baseline model. MLM-pretraining improves the model by providing a better initial representation of SMILES string 2. While the performance gap between the models narrowed over time during training, the MLM-pretrained model consistently achieved superior results across all evaluation metrics 1.Thus, incorporating MLM-pretraining proved beneficial for improving regression performance on this task.

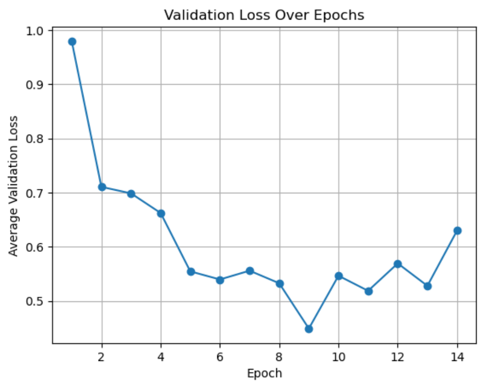


Figure 1: Validation loss across epochs for the Non-pretrained model; The model was selected after 9 epochs

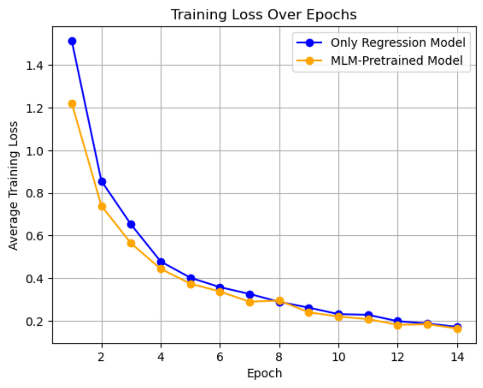


Figure 2: Training loss comparison across epochs for Non- and MLM-pretrained models

4 Influence Function-based Data Selection (Task 2)

In task 1 we saw the performance of an MLM-pretrained model compared to a simple regression model with respect to the Lipophilicity dataset. In this task there is an external dataset containing SMILES and label data, and the goal is to understand which of these data points actually improve the model's performance. This will be achieved by using influence functions(Koh and Liang, 2020) to compute the impact of each external data point on the model's behavior. We will select top-performing sample points based on influence score and include them in the training.

4.1 Model and Data Preparation

For this task, we will load the pretrained saved model from task 1 and use the relevant tokenizer for the model from Hugging Face(as the base model and dataset are from Hugging Face). We want to calculate the influence score of every sample point with respect to the test dataset, therefore we

Criterion	Original Model	MLM-pretrained Model
MSE	0.491021	0.443107
MAE	0.528994	0.497352
R^2-Score	0.655757	0.689348
Margin for Median	0.420027	0.384101

Table 1: Final evaluations of Non- and MLM-pretrained models

will load the main dataset and split the test set with a test size of 0.2(20% of the data). As we are dealing with molecular data, we will tokenize the SMILES string to the relevant format token so that the model can process that data. We are dealing with a regression problem here, hence we will define the Mean Squared Error (MSE) loss function to analyze the predictions.

4.2 Influence Score Computation

In this step influence scores are calculated. This is achieved as following:

- First calculate the test gradient for all test points and take an average. In this step make sure that neither any single gradient nor the average gradient is NaN or Infinity due to numerical instability in the calculation. If so replace them with zeros and normalizes the gradient vector by its norm to ensure stable scaling.
- The second step is to calculate the inverse Hessian-vector product. It is expensive operation-wise to compute the Inverse Hessian-vector product in deep neural networks. To address this, we use an approximation of the inverse Hessian-vector product (iHVP) using Stochastic estimation/LiSSA technique (Agarwal et al., 2017)
- In this last step we calculate the gradients for each point in external data through backpropagation and utilize the same techniques as explained in the first step to avoid any numerical instability. Now the influence score is calculated as the negative dot product of the iHVP and external data point gradient(whose influence score is being calculated w.r.to. the test dataset).

4.3 Data Selection

Once we calculate the influence scores, we can analyze these to understand the underlying pattern to make an informed decision in choosing top-performing points. As the influence score is a negative dot product, the highest negative influence score(if we sort the list in ascending format then the first element) is the best addition to the model's performance. Now question is which data points to be chosen for improving the model performance? To answer this we ran some basic statistical analysis on the derived influence scores. When we looked at the Histogram of the scores(Refer 3), we can see that data distribution is skewed left, meaning a long tail of highly negative scores. When we selected a threshold to choose lower endpoints with 33 percentile, we can see that the threshold is estimated at around mid-way to the whole distribution(-7.5 in this instance). The number of data points selected using this method is roughly about 1/3rd of the total external data points(which is about 100). We cannot choose the points based on standard deviation as we can see that distribution is not normal further using the Anderson-Darling Test we verify that this score pattern doesn't follow normal distribution.

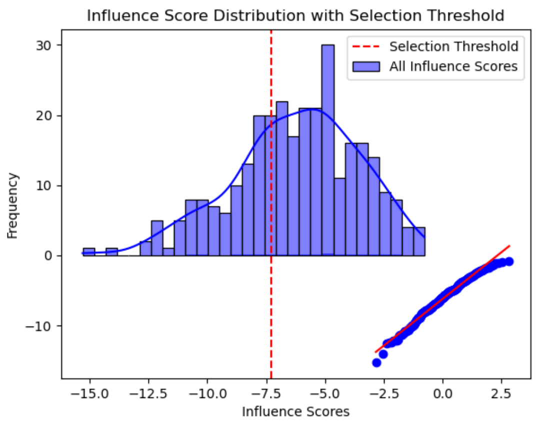
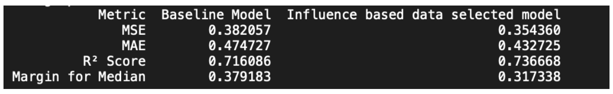


Figure 3: Influence Score Distribution with Selection Threshold

4.4 Training and Evaluation

We combined these selected data points with the original training split and created new tokens to train the pre-trained model from task 1. Evaluation of this trained model shows improvement in metrics like MSE, MAE, R^2 score, and Margin for median compared to the base model from Task 1 (Refer 4).



```
           Metric  Baseline Model  Influence based data selected model
              MSE        0.382057                             0.354360
              MAE        0.474727                             0.432725
         R² Score        0.716086                             0.736668
Margin for Median        0.379183                             0.317338
```

Figure 4: Comparison of Model Metrics

When we look at the plot of training loss over epochs(Refer 5) we can see that the pretrained Model consistently maintains lower loss throughout all 20 epochs, suggesting better overall performance during training. The Influence Function-based model shows more dramatic improvement in early epochs and at the end the gap between the models is very narrow. Analyzing the plot of Prediction Accuracy(Refer 6) shows fewer points outside the median margin for the Influence Function-based model. But overall the pre-training approach likely provides valuable initial knowledge that the influence function approach cannot fully compensate for, even with targeted data selection.

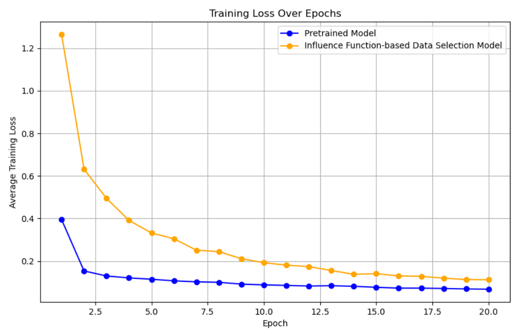


Figure 5: Training loss over epochs

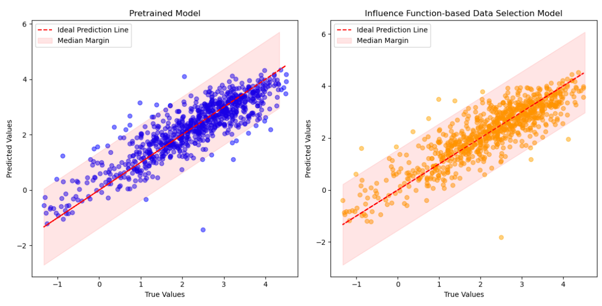


Figure 6: Prediction accuracy

5 Exploration of Data Selection and Fine-Tuning Methods (Task 3)

5.1 Data selection strategies

5.1.1 Choice of data selection strategy

For this section, we chose an **uncertainty-based data selection method**. Since this is a regression task, we use the **Monte-Carlo dropout method**.

The MC Dropout runs forward passes multiple times for each input and then collects predictions for each input in the dataset, generating a distribution of outputs to estimate uncertainty. Then we calculate uncertainty scores as the variance across MC predictions for each sample. Lastly, we select samples with the highest uncertainty scores.

We played around with the threshold based on which a sample was to be selected, trying threshold values from the 50th percentile up to the 90th percentile. We found that the the 90th percentile (top 10% of data samples with the highest of uncertainty scores) resulted in much lower MSE and MAE and a higher R^2 score.

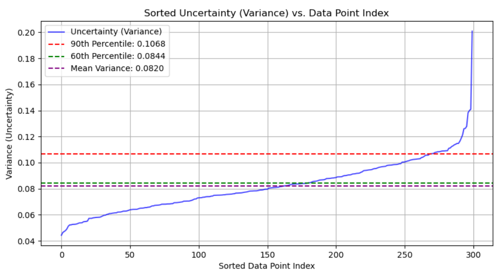


Figure 7: uncertainty scores of samples, arranged in increasing order of scores

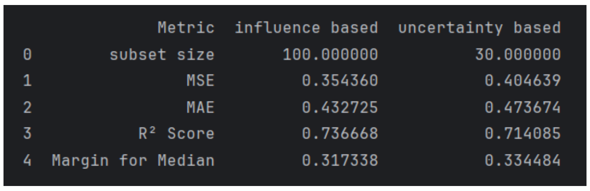
5.1.2 Comparison with influence-based data selection

Here is the comparison between the performance of the influence-based data selected and uncertainty based data selected datasets on the regression fine-tuning task:

As seen in the table above, the influence based data selection strategy has a data subset of size 100 and the uncertainty based data selection strategy has a data subset of size 30.

5.2 BitFit, LoRA and iA^3 PEFTs

All the 3 PEFT tasks were performed using the new dataset which contained selected sam-



```
               Metric  influence based  uncertainty based
0         subset size       100.000000          30.000000
1                 MSE         0.354360           0.404639
2                 MAE         0.432725           0.473674
3            R² Score         0.736668           0.714085
4   Margin for Median         0.317338           0.334484
```

Figure 8: comparison table

ples that were selected via the uncertainty based data selection method from task 3A.

5.2.1 BitFit

The BitFit implementation requires us to only train the biases in the model. For this model we use a simple "bitfit" flag in the MolFormer and adjust the training function to only train the biases and regression head.

Without any changes this PEFT gives a R^2 score of 0.41. However, we increased number of epochs from 20 to 40 and saw an improvement of R^2 score to 0.54. This was the best result from the BitFit PEFT we have seen.

5.2.2 LoRA

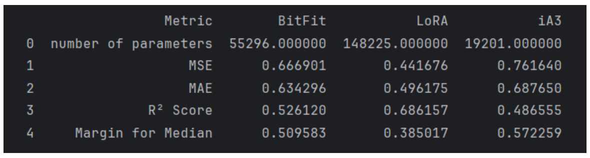
For LoRA we replaced the linear layers in the attention layer with special LoRA layers, that froze the original parameters and introduced the matrices B and A by means of the new LoRA class. The training function was updated to only train parameters that were not frozen.

LoRA without changes had a R^2 score of about 0.48 when we initially set rank=8 and alpha=16 as the authors of the original paper. We however had 4 hyperparameters we could tune - rank, alpha, number of epochs and learning rate. We see that LoRA performs very close to the actual full fine-tuned regression task with rank=4, alpha=64, epochs=40 and learning rate=$2e^-5$. More precisely, LoRA had a R^2 score = 0.69 with roughly 150,000 parameters (in comparison to R^2=0.72 for 44 million parameters in the full fine-tuned version).

For reference, we tried rank values = [2,4,8,16,32] and alpha values = [4,8,16,32,64]. We adjusted the learning rate on the go.

5.2.3 iA^3

For iA3Transformer we created a new class that had the scaling vectors for the attention layer in each layer of the model. The iA^3 computation was done directly in the new forward function. The training function was adjusted to only train parameters that were not frozen.



```
              Metric       BitFit           LoRA           iA3
0  number of parameters  55296.000000  148225.000000  19201.000000
1                   MSE      0.666901       0.441676      0.761640
2                   MAE      0.634296       0.496175      0.687650
3              R² Score      0.526120       0.686157      0.486555
4     Margin for Median      0.509583       0.385017      0.572259
```

Figure 9: PEFT performances

For iA^3 we could only play around with epochs, learning rate and the value to which the scaling vectors were initialised. The authors originally initialised the scaling vectors to 1, however, we found that the model was performing better when initialised to 0.8 instead. The epochs didnt make much of a difference however the model seems very sensitive to the learning rate. We found the best results at lr=$1e^-3$

The results for iA^3 seem to be a little underwhelming, with the R^2 only being as good as 0.48. Further hyperparameter tuning hasnt helping in improvements.

References

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

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