Delft University of Technology

Application of Language Models to homogeneous catalysis

Final Report

Master thesis

Authors:
Jan de Korte

Daily Supervisor:
A.V. Kalikadien
Responsible Supervisor:
Prof. Dr. E. A. Pidko

July 12, 2024

Abstract

Asymmetric hydrogenation is a field of major interest for the pharmaceutical industry. Using these catalyzed reactions instead of traditional stoichiometric reactions can reduce waste, energy and can open up possibilities to new intermediates, products and synthesis pathways. Finding an optimal catalyst to produce a selected enantiomer remains a struggle however, requiring large time and resource investments. Determining ligand performance can be done experimentally using HTE campaigns, supplemented with predictive methods, either mechanism-based or mechanism-agnostic. Recent advancements in the mechanism-agnostic predictive methods include a large range of studies using Machine Learning approaches, relying mostly on using molecular descriptors to represent the catalyst structure to the models. More recently, with the rise of NLP models, string-based structural identifiers are used to train a Language Model to predict catalyst performance. In a recent study used an LSTM model was trained and used to predict the enantiomeric excess of a range of ligands for an asymmetric hydrogenation reaction. This work is based on the workflow used by them, and validates the performance of this model as showed in their paper. Furthermore, this model was applied and tuned to predict the enatiomeric excess of a range of ligands, based on a dataset from the ISE research group.

Contents

1. Introduction

With the global overuse of natural resources and overproduction of waste, endangering the future of our planet, numerous attempts are made to improve the resource efficiency in industries [1]. A well-known example is the oil refining industry, where continuous improvement in process conditions among others have increased the efficiency of the overall process. Table 1.1 shows a comparison of this industry with other large chemical industry sectors. This shows that the oil-refining industry is indeed relatively resource-effective, especially compared to the pharmaceutical industry.

Industry sector	Tonnage [ton]	E Factor [kg waste/kg product]
Oil refining	$10^6 - 10^8$	< 0.1
Bulk chemicals	$10^6 - 10^8$	$< 1 - 5$
Fine chemicals	$10^6 - 10^8$	5 to > 50
Pharmaceuticals	$10^6 - 10^8$	25 to > 100

Table 1.1: E Factors in the chemical industry [2].

Part of this difference can be attributed to the smaller scale and batch-production nature of this industry sector. A larger scope of different products in smaller quantities are produced. However, a major difference between the processes in these sectors is the absence or scarcity of (optimized) catalysts [1]. While most reactions in the oil refining industry are catalyzed by relatively highly optimized and well-studied catalysts, for the pharmaceutical industry this is not the case. One reason for this is the aforementioned smaller-scale production, decreasing the feasibility of a thorough search for an optimal catalyst[3].

This shows that discovering and developing catalyst for the pharmaceutical industry can increase the resource-efficiency and sustainability of this industry sector. Using a catalyst can not only reduce the use of energy and resources but ultimately reduce waste production, by increasing selectivity and lowering activation energy. Especially increasing the selectivity towards the desired product is of interest for the pharmaceutical industry. Since catalysts can not only improve chemeoselectivity, but also also diastereoselectivity and enantioselectivity (Figure 1.1), the unwanted production of by-products from the reaction in stoichiometric balance can be largely reduced [4].

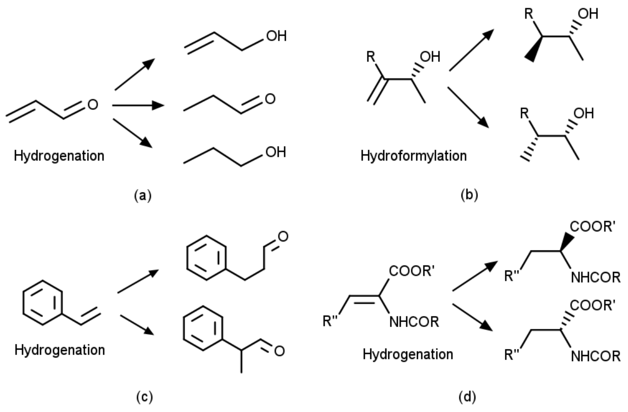


Figure 1.1: Examples of selectivity for chemical conversions [5]. (a) Chemoselectivity, (b) Regioselectivity, (c) Diastereoselectivity, (d) Enantioselectivity

These properties are of high interest for the pharmaceutical industry, as often only one enantiomer is of interest for application. Especially for hydrogenation reactions, where in the pharmaceutical industry only one enantiomer is of interest. Hydrogenation reactions play a large role in the pharmaceutical industry [6]. While there are methods to separate the produced enantiomers based on their chirality, asymmetric hydrogenation, using a chiral ligand, can produce enantiopure compounds [7]. Asymmetric hydrogenation thus not only yields higher selectivity, but with that also reduces the downstream process complexity [5]. Asymmetric hydrogenation focuses on homogeneous catalysts, since while biocatalysts are known to have enantioselective properties, and although the field of biocatalysis has been rapidly evolving, quite a lot of challenges still complicate the application [1][8].

So, with this great potential, the challenge is to optimize the search for an optimal catalyst, to improve the development of catalyst for use in the pharmaceutical industry [9]. Traditional catalyst research is mainly focused on experimental research around catalysts that are known to have good catalytic properties, further guided by chemical intuition of the researcher. Efforts have been made to improve the research process, with use and development of high-throughput experiments (HTE) have improved the speed and cost at which research is performed[10].

The workflow of HTE campaigns can be further improved by using computational tools and methods. Mechanism-based predictive methods aim to predict suitable catalyst candidates using quantum chemical calculations of key transition state intermediates. These methods rely on understanding and knowledge of the reaction mechanism.

Predictive methods that do not require deep knowledge of the mechanism rely use data-based approaches. These methods aim to predict candidate catalysts by finding a relation between the structure of the catalyst and performance indicating. The use of data-based approaches has two major advantages over mechanism-based methods [11]:

1. Catalyst design is accessible to more researchers, since for the mechanism does not need to be deeply understood to use the tools available.
2. Predictive models can be easier transferred between catalytic systems.

Because of these advantages, this work focuses on data-based approaches, also called Quantitative Structure-Property Relationship (QSPR). In these methods, the candidate-catalyst structure is generally digitally represented by structural descriptors, and form a dataset together with experimental data on performance indicators. This dataset can then be used to find correlations, and make a model to predict those indicators on other candidate-catalysts. This model is often based on a regression model or a Machine Learning (ML) model, since the relationships are often non-linear.

More recently, with the development of Natural Language Processing (NLP) models, the need to calculate the descriptors is removed, as this opens up opportunities to use textual identifiers, as SMILES or InChI, to represent the structures digitally. NLP models are neural networks, which are able to 'understand' semantics and relationships between words in a language, and make predictions based on that. These models are applied to tasks as translating, summarizing, and classifying text.

When NLP models are used in QSPR approaches, the language to learn are chemical textual identifiers, as SMILES or InChI. After training the model, it can be used to predict candidate catalysts by classification or predicting the performance of the catalyst. In recent years, a number of studies have used NLP models, although with a range of different approaches.

Jablonka et al. showed the performance of a Large Language Model, GPT-3, which was trained on English text, in relating molecular structures to properties [12]. With examining a range of different target tasks, as classification, regression and inverse design, they show the large potential and ability of an LLM to use learned logic on new datasets, both in fine-tuning and in-context learning. This is supported by other studies, mentioning the 'inherent chemical knowledge' of LLM's [13]. The prediction tasks were performed of a range of different datasets, including prediction and classification of transition wavelengths of photoswitches, HOMO-LUMO gaps and solubility. Furthermore, performance of yield prediction of Pd-catalysed Buchwald-Hartig C-N crosscoupling [14] and Pd-catalysed Suziki-Miyaura C-C cross-couplings [15] was shown.

Yoshikai et al. showed the potential of Transformer models in predicting molecular properties from molecular descriptors and SMILES data [16]. One of their main findings was that Transformer models in particular can have difficulties with predicting properties of chiral molecules. The performance of the model was explored with a range of datasets, often used as benchmark datasets for ML models, such as BBBP [17]), FreeSolv [18] and Lipophilicity [19].

Li and Fourches in 2020 and later Singh and Sunoj in 2022 thoroughly examined performance of LSTM models in a broad range of scenarios [20],[21]. While Li and Fourches showed the potential of this model with a few different datasets, focusing on the effect of data augmentation, Singh and Sunoj showed the potential of using relatively small datasets, again using data augmentation. Most notably, Singh and Sunoj reported good performance for a model trained to predict the enantiomeric excess of a range of ligands for hydrogenation reactions.

The promising results of Singh and Sunoj, their overlap in application on asymmetric hydrogenation and the use of a relatively small dataset and model inspired me to look deeper into application of NLP models on asymmetric hydrogenation, following the workflow of Singh and Sunoj. Since their workflow was developed by Li and Fourches [20], both studies will be mentioned frequently. In this work, I aim to answer the following question: **How does the existing workflow to a fine-tuned language model, developed by Li and Fourches, perform on a new, unseen dataset for enantioselective hydrogenation?** This question will be answered by looking deeper into the following subquestions:

1. To what extent can the results of the study of Singh and Sunoj, using this workflow, be recreated?
2. To what extent are the results of the study of Singh and Sunoj reproducible using a new version of FastAI?
3. How do the datasets of Singh and Sunoj compare to the dataset used in this study?
4. How can the existing workflow be applied to the dataset used in this study?
5. How can this workflow be further optimized to achieve better performance for the language model for the dataset used in this study?

2. Theory

2.1 Homogeneous Catalysis

As mentioned, the study of homogeneous catalysis has great potential, because of the relatively high selectivity of the catalysts. This is true, especially compared to heterogeneous catalysis. This section aims to take a deeper look into homogeneous catalysis, asymmetric hydrogenation in particular, and explain the concept of enantiomeric excess.

As mentioned, homogeneous catalysis is, among other advantages, generally more selective [22]. While acids and enzymes are classified as homogeneous catalysts, the most prominent group is the transition metal catalysts. These catalysts consists of a transition metal, surrounded by one or more ligands. While the catalytic properties of these complexes originate from the available oxidation states of the transition metal center, the ligands play a large role in determining the catalytic activity [5]. There is a wide range of ligands available, classified as mono- and polydentate ligands. These chelating ligands offer additional stability compared to monodentate ligands.

For asymmetric hydrogenation, Knowles and Horner introduced Rhodium-based catalysts [23][24]. The catalysts used for asymmetric hydrogenation consists of asymmetric, chiral ligands, which result in good enantiomeric excess and a 100% theoretical yield [25][26]. Enantiomeric excess is a metric to determine the extent in which an asymmetric hydrogenation reaction produces enantiopure products. It is calculated as

$$|ee| = |F_R - F_S| \tag{2.1}$$

Where F_R is the mole fraction of R isomer and F_S is the mole fraction of S isomer, so that $F_R + F_S = 1$. Other ways of expressing enantiomeric excess include the true value of Equation 2.1, that is, ee, not the absolute value, and expressing the enatiomeric excess in percentages, denoted $\%ee$.

2.2 Natural Language Processing

2.2.1 Language Models

Language models consist generally of 3 major parts, the embedding layer, the encoder, and the decoder. These, together with a tokenizer and a dataloader are the core of the NLP workflow. This section explain the functions each part has, and discusses the relevant options and choices that can influence the result.

Tokenizer

For the input to be used in the model, the SMILES data has to be converted to numerical values. This is done in a few steps, starting with splitting the input data into into smaller, repeatedly occurring, parts, called tokens. Tokenization can be done on character-level, but also larger parts of the input can be used, to increase the amount of information provided per token, making it easier for the model to learn [27].

For textual data, these options are character-level, word-level or subword-level tokenization. Using character-level tokenization will result in a small vocabulary (unique tokens in data), but losing relationships and semantics [28]. However, word-level tokenization, although increasing the information provided with each token results in a very large vocabulary, increasing the computational and memory requirements. The best of both worlds is found in subword tokenization, using Byte Pair Encoding (BPE) to create a vocabulary of the most frequent occurring subwords [27].

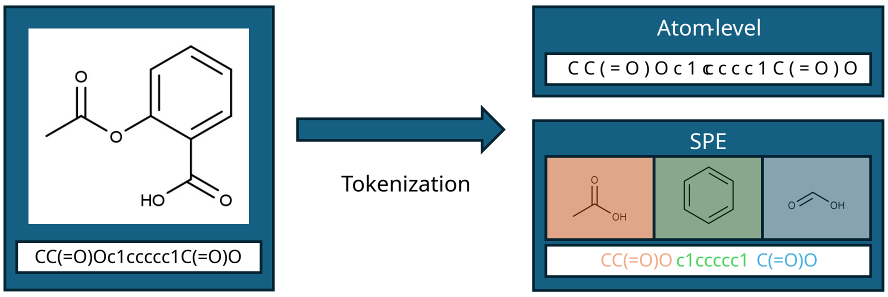


Figure 2.1: Example of tokenization, showing the difference between atom-level and SPE tokenization [20].

Li and Fourches researched the effect of using SMILES Pair Encoding (SPE) on SMILES data. An example is shown in Figure 2.1. It was found that the performance of a model can be increased significantly when using SPE instead of character-level (or atom-level) tokenization [29]. When using atom-level tokenization, common practice is to include a set of 'special characters' which should not be tokenized on atom-level [20]. These often include 'Cl', which should not be tokenized to 'C l', as well as chiral information as [C@@H] and [C@H].

Dataloader

A dataloader is an essential component that handles most of the processing of data for a NLP model. It performs several tasks to efficiently process the data, and use the data efficiently during training. A dataloader preforms tasks as normalization, where the data is padded to use in Tensors, shuffling the data during learning to avoid overfitting, and allow for parallel loading of data.

Depending on the tools used, a dataloader can also apply efficient tokenization and numericalization. Since an NLP model is a computational model, text has to be transformed to numerical data to use in the model. This is done by converting each token, as provided by the tokenizer, to an vocabulary index.

Embedding layer

The numericalized data has to be processed further before it can be processed by the NLP model. While the input data is now long arrays of numerical data, NLP models work better with high-dimensional vectors of a fixed size. This can be done by one-hot encoding, where each token is represented by a vector with the length of the vocabulary. For larger datasets, which require larger vocabularies, these one-hot encoded vectors, which are sparse by definition, are transformed to more dense vectors. These vectors, either dense or sparse, are the embedding vectors [30].

During the training of the model, the embeddings are updated by backpropagation to minimize the loss function. With this process the embedding layer learns to capture properties of the data better[30].

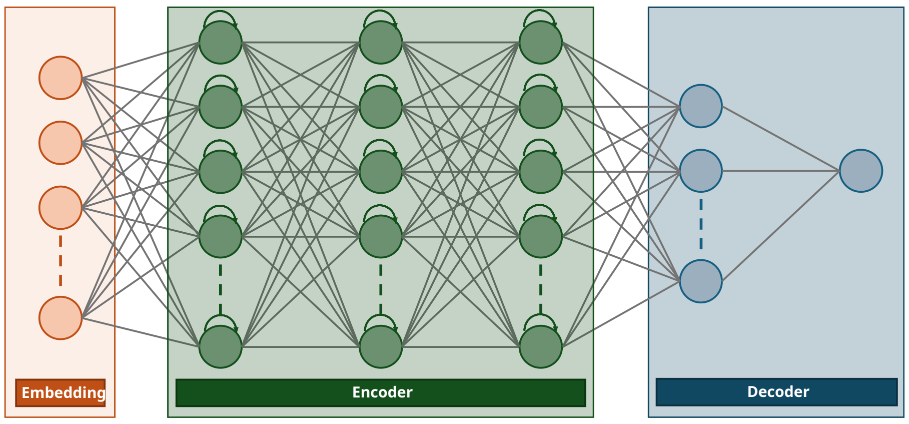


Figure 2.2: Schematic diagram of a general RNN architecture.

Encoder

The function of the encoder layers is to grasp relationships between tokens, generating a contextualized embedding to pass to a decoder. This process provides a sense of context and dependency between the tokens, giving a better understanding of the meaning of the token. The output of the encoder is again embedded vectors, but containing contextual information [31].

Decoder

Since the output of the encoder is a dense vector containing contextual information for each token, this can be used to generate predictions. These predictions are dependent on the target task, and can vary from sequences, like text in a translator, to integers, like classes in a classifier, to floats, like values in a regressor. The decoder can consist of multiple layers, and can be, depending on the architecture, linear, for example in an LSTM architecture, or more sophisticated, for instance, with attention layers in a Transformer architecture.

2.2.2 Architectures

For NLP models a few different architectures are available. In this section, the Recurrent Neural Network and Transformer architecture will be further investigated.

Recurrent Neural Network

Recurrent Neural Network is a neural network architecture that is designed for processing sequential data. This architecture consists of layers of neurons that process the data. Since this architecture is recurrent, the output of the layers is not only dependent on the current input, but also on previous inputs. This makes the model capable of capturing temporal dynamics. Aspects of previous inputs are stored in a short-term memory, or hidden state [32].

Some downsides of this architecture are the risk of over- and underfitting of the model, due to extremely high or low error rate gradients. Furthermore, while the architecture is designed for processing text, it does this sequentially, making it less fit for training on databases with large sets of data, such as articles [33].

Long Short-Term Memory

To address those issues, LSTM architecture was designed. This Long Short-Term Memory encoder is more capable of capturing relations between and pattern in tokens further away from each other. This overcomes the problems with underfitting due to very low error rate gradients [32]. In the LSTM architecture, LSTM cells are added to the hidden layer, capturing information from previous inputs. However, LSTM architecture still has some of the disadvantages of the general RRN architecture, as the sequential aspect of data processing remains, with the increased memory and computational requirements for long training data. Furthermore, long sequences with long-distance dependencies between tokens are still difficult for the model to grasp [34].

Transformer

This was addressed by the development of the Transformer architecture, which uses the concept of 'attention' and parallel processing of data to relate input tokens to each other, regardless of the distance between those tokens. The Transformer architecture adds an attention layer to the encoder, as well as the decoder. A multi-head attention layer can employ attention to several important parts of the input. Since niput data is processed in parallel, the Transformer architecture is more capable of processing long input sequences, while doing this also more resource-efficient [33].
However, due to the recurrent nature of LSTM models, and the larger number of parameters of Transformer models, on smaller datasets LSTM models achieve higher performance. This advantage is further expanded by the fact that the memory requirement of LSTM models scales linearly with sequence length, compared to quadratically for Transformer models [35].

2.2.3 Training

To actually apply a NLP model on data and a target task, the model has to be trained. There are various approaches to arrive at a model ready for inference, however, only the Transfer Learning approach with ULMFiT is described.

Transfer Learning

Training a NLP model is computationally expensive, and requires a large dataset, a search for the best architecture and tuning architecture parameters to arrive at the optimal model for the target task. With the use of transfer learning, the amount of data required can be reduced significantly, reducing the computational costs as well. In transfer learning, the model is trained on a large dataset. This enables the decoder to learn to contextualize the data, updating the weights of the decoder layers. This training, pretraining, is done on a 'source task', like predicting the next word in a sequence [36].

When applying transfer learning, the weights of the decoder are transferred, while applying a new 'head' or encoder. This enables the model to be trained on a much smaller, although closely related, dataset, where especially the encoder is trained to the target task [37].

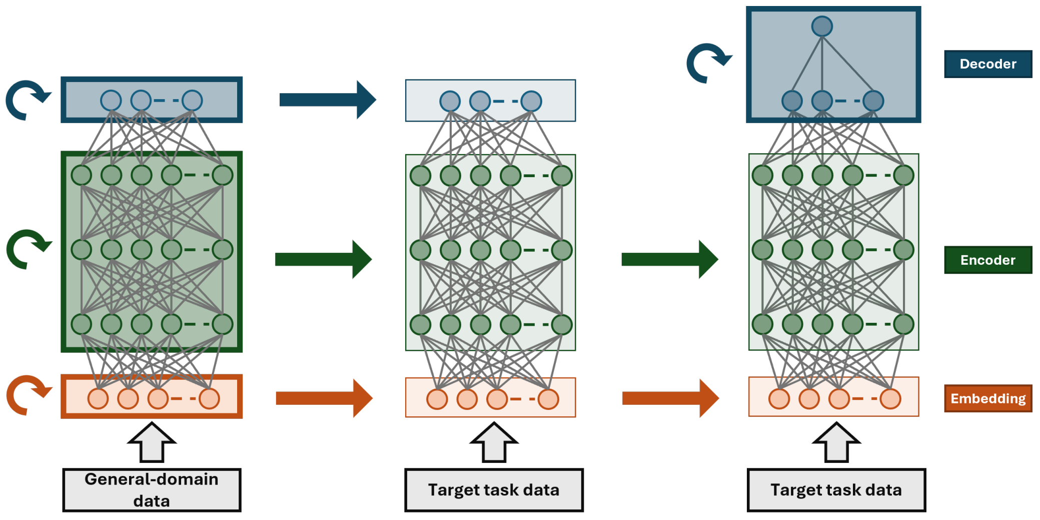


Figure 2.3: Overview of the three stages in transfer learning approach. Left: General-domain LM pretraining. The model has to be trained from scratch since all weights are randomly initialized. Middle: Target task LM fine-tuning. All weights from the pre-trained LM are transferred. Right: Target task classifier fine-tuning. The embedded and encoder weights are transferred, while the decoder is initialized and need to be trained from scratch.

Transfer learning for NLP models, especially focused on classification, is often done using the Universal Language Model Fine-Tuning (UMLFiT) approach. The workflow of this approach consists in general of three phases [37], and is visualized in Figure 2.3:

1. **General domain LM pre-training** An extensive dataset, representative of the language used for the target task, is used to train the NLP model. This allows the NLP model to capture general features of the language. Due to the large dataset and since the model has to be trained from scratch, this step is the most expensive. However, due to the transfer learning approach, this has only to be done once. This step is shown in Figure 2.3, the left part.

2. **Target task LM fine-tuning** The NLP model is then trained on the target data, using sophisticated training schedules. This 'fine-tuning' benefits from the pre-training, and is aimed at grasping the features specific to the target data. The use of pre-trained weights and sophisticated training schedules allows to fine-tune on a much smaller dataset, with less risk of overfitting. This step is shown in Figure 2.3, the middle part.

3. **Target task classifier fine-tuning** Finally, to train the NLP model on the target task, the encoder is replaced. This means that in fine-tuning only the encoder weights are randomly initiated, and trained from scratch. Again, sophisticated training schedules ensure the encoder is trained rigorously enough, while ensuring more careful learning on the decoder layers. This step is shown in Figure 2.3, the right part.

Fine-tune schedules

As mentioned, in Target task classification fine-tuning, different layers should be trained with different levels of aggression. However, the same is true for Target task LM fine-tuning, as features are captured over multiple layers [38]. There is a broad range of fine-tuning options available, some of them will be discussed here.

Since not all layers should be tuned to the same extent, '*discriminative fine-tuning*' was proposed [37]. This method splits parameters of the model, so that to each layer a different learning rate can be applied. In general, the learning rate of the last layer should be higher than the lower layers. Furthermore, it is proposed to let learning rates vary during training [37]. This approach aims to obtain a quick convergence in a general direction, with a higher learning rate, after which the parameters are refined using a gradually decreasing learning rate. This yields a triangular learning rate over time, hence the name '*slanted triangular learning rates*'. Last, to avoid catastrophic forgetting by tuning all layers at once, '*gradual unfreezing*' training schedule was proposed [37]. In this schedule, all layers are 'frozen', except the last. After fine-tuning, the last frozen layer is unfrozen, and fine-tuned. This is repeated, until all layers are tuned.

Performance metrics

To indicate the performance of the model, a few different metrics can be used. First there is the result of the loss function for both the training sample and the validation sample. Both give an indication of how well the model captures the training data and performs on the validation data respectively. The difference between those two performance indicators provide a indication of whether overfitting occurs. A high validation loss, with a low training loss shows the model is capable of performing the target task for the training data well, but fails to generalize this to the validation data, which is a sign of overfitting. Other parameters used for regression target task can be Root Mean Squared Error (RMSE), Mean Absolute Error (MAE) and R^2. For a Language Model learner, this is often accuracy and perplexity, denoting the ratio of correct predictions and a measure of the correctness of a sequence of words respectively.

Hyperparameters

Efficiency and performance are two important metrics of an NLP model. These can be optimized by tuning of hyperparameters. Hyperparameters of an NLP model are architecture-related, like number of hidden layers, size of hidden layers, but also training-related, like learning rate and number of training epochs. The optimal set of hyperparameters change with dataset and target task. Table 2.1 shows an overview of selected hyperparameters and their influence on performance or efficiency.

Training hyperparameters	
Learning rate	Speed, stability
Batch size	Speed, efficiency
Number of Epochs	Prevents under- and overfitting
Dropout rate	Prevent overfitting
Momentums	Speed, stability
Weigth decay	Prevent overfitting, stability
Architecture hyperparameters	
Number of hidden layers	Performance, efficiency, prevent underfitting
Size of hidden layers	Prevent under- and overfitting, efficiency
Activation functions	Performance, stability
Optimizer	Speed, stability
Sequence length	Performance, efficiency, stability, prevent overfitting
Embedding size	Performance, efficiency

Table 2.1: Overview of key hyperparameters and their effect. Adapted from [39]

2.3 A transfer learning protocol for chemical catalysis using a recurrent neural network adapted from natural language processing [21]

As mentioned, Li and Fourches [20], and later Singh and Sunoj [21] explored the possibilities of the LSTM model. Li and Fourches showed the effect of data augmentation. They furthermore benchmarked the data with existing studies on well-known datasets. Using the workflow from Li and Fourches, Singh and Sunoj showed the influence of LM fine-tuning on target task and also benchmarking the performance with existing models. They furthermore use the data augmentation workflow, developed by Li and Fourches, to show that this allows training with relatively small datasets.

2.3.1 Datasets

In the study of Singh and Sunoj, three datasets were used, all containing enantiomeric excess and yield of catalytic reactions. Figure 2.4 gives an overview of the datasets used, and was provided in the published paper.

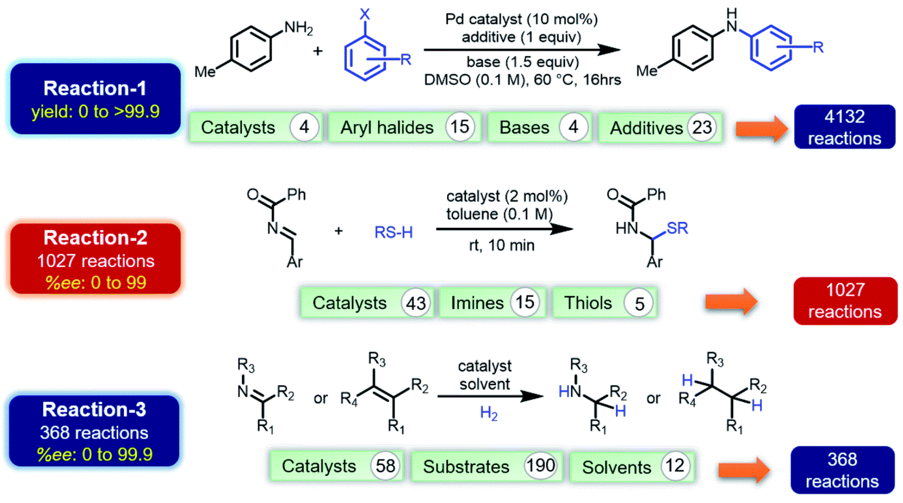


Figure 2.4: An overview of the reactions and datasets chosen, as provided by [21]

The first dataset consists of a range of Buchwald-Hartig reactions (Reaction 1). With varying catalysts, acryl halides, bases and additives, this datasets contains 4132 datapoints. This reaction is widely applied in the pharmaceutical industry. The second dataset contains a range of enantioselective *N,S*-acetal formation reactions (Reaction 2). This dataset consists of 1027 datapoints, a combination of catalysts, imines and thiols used, with for each combination their enantiomeric excess. The last dataset is a range of 368 asymmetric hydrogenation reactions (Reaction 3). These datapoints contain enantiomeric excess values for a combination of catalysts, substrates and solvents.

Furthermore, a dataset of 1 million SMILES strings was used to pre-train a language model.

2.3.2 Architecture

The model used in the study of Singh and Sunoj is an AWD-LSTM model, Non-monotonically Triggered Average Stochastic Gradient Descent-LSTM model, employing additional regulerization and optimization techniques on top of the LSTM model as explained in Section 2.2.2. Looking at Figure 2.2, the different sizes of the layers can be specified. For the regressor, the size of these layers are shown in Figure 2.5.

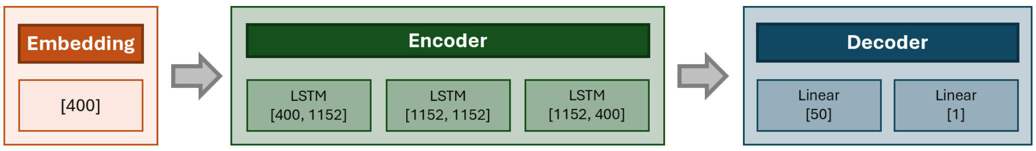


Figure 2.5: Architecture of AWD-LSTM model used in the study of Singh and Sunoj. Adapted from [21]

2.3.3 ULMFiT

These datasets are used in ULMFit training approach. For each dataset, three approached were used. One with the true ULMFiT approach, using the pre-trained model, fine-tuning the Language Model on the target data, then fine-tuning the model on the target task (TL-m2). Furthermore, to investigate the effect of fine-tuning the LM, a study was performed without fine-tuning of the LM (TL-m1). Last, to show the effect of transfer learning, a study was done without using pre-trained weights (TL-m0). The results of this comparison show little to no effect in using transfer learning (comparison of TL-m0 and TL-m1) for Reaction 1, with a large dataset. However, for reaction 2 and 3, there was a noticeable improvement in performance, giving a decrease in RMSE of 11.83 to 8.88 and 10.67 to 8.56 respectively.

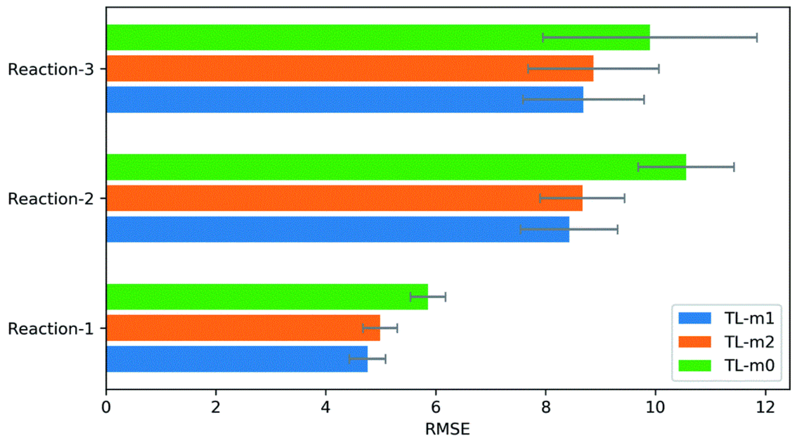


Figure 2.6: Comparison of performance of direct (TL-m0) and transfer learning methods (TL-m1 and TL-m2), as provided by [21]

Using LM fine-tuning showed interestingly enough a decrease in model performance. Although it is small, it is consistent for all three reaction datasets. Figure 2.6 shows an overview of the performance found.

The most important take-aways from this study are

- Transfer learning has a positive effect on performance for relatively small datasets
- Fine-tuning the LM on target data is not always beneficial for performance
- Gradual unfreezing while fine-tuning is not always the best approach to reach optimal performance.

In the end, the workflow used in this study provides a useful starting point for our study. Be informed that this workflow was not developed in this study, but in a previous study by Li and Fourches [20].

3. Methodology

This section will explain the workflow and research steps of this project, with references back to relevant sections in the Theory chapter. This section is divided in three subsection, where first the data preparation and processing is described, followed by a description of the approach to fine-tune the NLP model, and finally explains how the performance of this model was examined.

3.1 Computational tools

Since this project has a heavy focus on computational tools, this section will explain which tools were used.

3.1.1 Environments

Google Colab

For a large part, Jupyter Notebook on Google Colab was used as an environment to perform the experiments. FastAI is installed by default, as well as a large number of other useful Python packages. Furthermore, to a certain extent, free runtime on Tesla T4 GPU's [40] is available, making this an easy and accessible way to perform experiments.

Snellius

To speedup experiments further, Dutch national supercomputer Snellius is used [41], making use of the GPU partition with Tesla A100 GPU's [42].

DelftBlue

Additional computation time was obtained on the computational cluster from TU Delft, DelftBlue, to make use of the Tesla V100 and Tesla A100 GPU's [43].

3.1.2 Python package FastAI

This work uses the Python package FastAI to set up the experiments. It consists of a few API layers, built on top of Pytorch. This makes the package easy to use, with the high-level API, while still allowing for modifications if required with the lower-level API. The functionalities are all well-documented, and explained in courses. Figure 3.1 shows the general structure of this package.

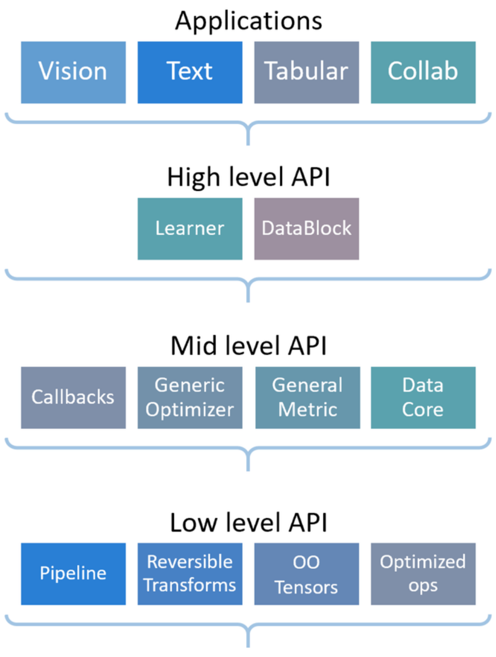


Figure 3.1: Functionality of FastAI layered in API-levels [44]

The functionalities used are a combination of the high-level and mid-level API, to allow for optimization of the workflow and model performance.

3.2 Data Processing

The data was prepared from the data from a previous study of the TuDelft ISE research group [9]. The data to be used was extracted from this dataset, formatting this in a .csv file, with 12 columns. The columns, and variation in content are shown in 3.1. The ligand indicators (ligand, ligand_int), ligand represented in isomeric SMILES (ligand_smiles), the substrate indicator (substrate), solvent indicator(solvent) and enantiomeric excess values, non-absolute and absolute respectively (ee, ee_abs) and conversion value (conversion), were provided in the dataset mentioned. To this, the ligand class (ligand_class) and ligand family name (ligand_family) were added, as well as substrate SMILES and solvent SMILES, obtained from the PubChem database [45].

Column name	**Number of unique entries**
ligand	192
ligand_int	192
ligand_smiles	192
ligand_class	8
ligand_family	75
substrate	4
substrate_smiles	4
solvent	2
solvent_smiles	2
ee	continuous [-1, 1]
ee_abs	continuous [0, 1]
conversion	continuous [0, 1]

Table 3.1: Overview of the prepared dataset

Furthermore, as mentioned, the data was split in a train-valid-test split with ratio 0.8/0.1/0.2. However, since the data contains datapoints with different substrates and solvents for the same ligand, this can cause overlap in training and valid/test data. The training data is then very similar to the validation and test data, which can skew the performance of the model unfairly positive. To overcome this, the data was first split by ligand, and then this split was used to split the rest of the data. This keeps certain ligands unseen for the model in training.

3.2.1 Data augmentation

As mentioned in Section 2.3, data augmentation seems a useful tool to improve the performance of the model. This method is often used in vision learning, where data is often rather scare. This method extends the dataset with data very similar to the original, but with added noise. This provides the model with a large amount of data, very similar, but still different. To prevent overfitting, but still improve the performance of the model, the right amount of noise should be selected. In NLP this method is often much less easy to apply, as this requires paraphrasing, random deletion or using synonyms. However, when using SMILES, this becomes much easier, as one structure can be represented by multiple SMILES strings. Together with adding noise to the target data, this can extend the dataset rather easily.
To implement this, the Chem package from the Python package RDKit is used. A SMILES string is converted to a molecule-instance from the mol-class. This molecule-instance is then renumbered, and converted back to SMILES. This gives a new SMILES string, representing the same molecular structure.

3.2.2 Tokenization and numericalization

As explained in section 2.2.3, textual data needs to be formatted in a way this model can understand. First, since the performance of the reaction is not only influenced by the ligand, but also the substrate and the solvent, these structures should form the input together. This is done by concatenating the SMILES data together, separated by periods. See for example Figure 3.2.

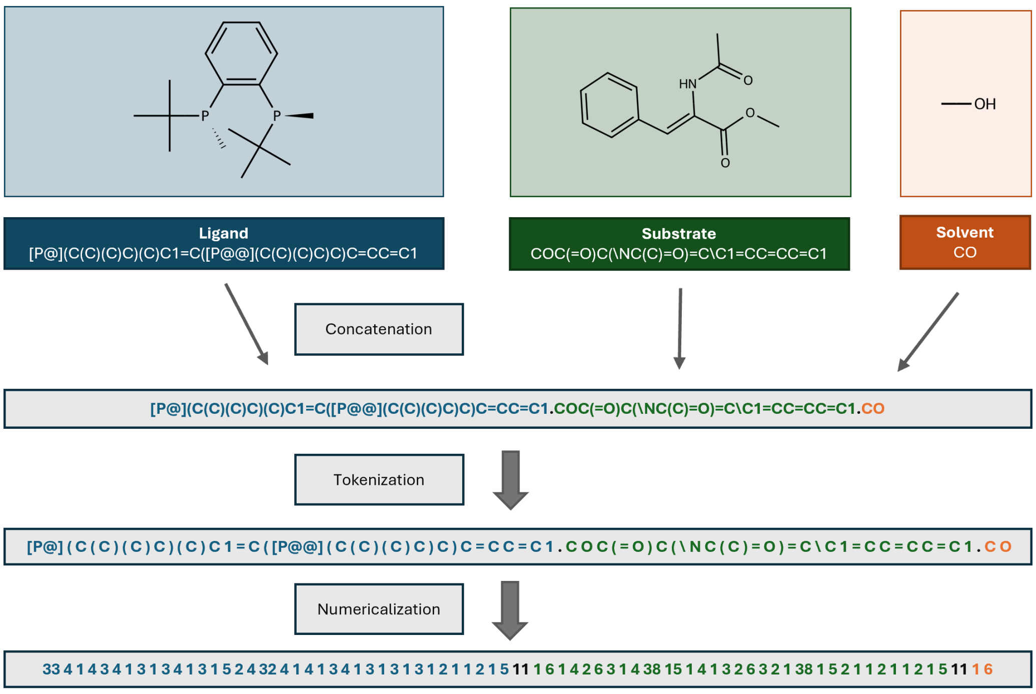

HN
O
O
O
P
P
—OH
Ligand
[P@](C(C)(C)C)(C)C1=C([P@@](C(C)(C)C)C)C=CC=C1
Substrate
COC(=O)C(\NC(C)=O)=C\C1=CC=CC=C1
Solvent
CO
Concatenation
[P@](C(C)(C)C)(C)C1=C([P@@](C(C)(C)C)C)C=CC=C1.COC(=O)C(\NC(C)=O)=C\C1=CC=CC=C1.CO
Tokenization
[P@] (C (C) (C) C) (C) C 1 = C ([P@@] (C (C) (C) C) C) C = C C = C 1 . C O C (= O) C (\ N C (C) = O) = C \ C 1 = C C = C C = C 1 . C O
Numericalization
33 4 1 4 3 4 1 3 1 3 4 1 3 1 5 2 4 32 4 1 4 1 3 4 1 3 1 3 1 3 1 2 1 1 2 1 5 11 1 6 1 4 2 6 3 1 4 38 15 1 4 1 3 2 6 3 2 1 38 1 5 2 1 1 2 1 1 2 1 5 11 1 6

Figure 3.2: Overview of the process of concatenating, tokenization and numericalization of SMILES data, with a structural representation of the selected SMILES of ligand, substrate and solvent

As mentioned in Section2.2.1, the data has to be numericalized. Since a transfer learning protocol was used, the data should be embedded in a similar manner, which means the tokenization and numericalization should be done too with the same tokenizer and a similar vocabulary. The tokenizer used was adapted from the work of Li and Fourches [29].

3.3 Implementation with FastAI

As mentioned, after setting up the tokenizer, the next step is to fine-tune a language model on the target task. This is an essential part of the ULMFiT approach, as this adapts the pretrained and randomly initialized weights for the new vocabulary entries in the embedding layer to the differences between the dataset used for pretraining and the target data.

3.4 Model performance

To reach optimal model performance, the training hyperparameters can be tuned. The training hyperparameters are mentioned and explained in section 2.2.3. Since this is a large set, and the optimal setting for one hyperparameter can influence the optimal setting for another, this is done in order of magnitude of influence on the performance, as found by Liao et al. [39]. Furthermore, in the search for the optimal hyperparameter, some rules of thumb were used [39], such as, keep the batch size a power of 2, increasing momentums works best with decreasing learning rates. Furthermore, the performance was not only measured by calculating the RMSE, MAE and R2-score of the test set, but also by monitoring the train and valid loss, and the difference between those, to keep the amount of overfitting as low as possible.

3.5 Analysis

3.5.1 Principal Component Analysis

To visually compare the similarity of the ligands in the dataset of Singh and Sunoj to the ligands in the ISE dataset, a Principal Component Analysis is used. This statistical method reduced the dimensionality of a large dataset to a number of principal components. These components are vectors through the dimensions of the data. The best principal components are the ones that explain the variance of the dataset best. For PCA, often the first two that explain the largest amount of variance are chosen, from which a 2D scatterplot is created. The principal components in this case are calculated from structural descriptors, calculated from SMILES strings of the ligands, using the Mordred Python package.

3.5.2 Receiver Operating Characteristic curve

To show the extent to which a classification model performs, a ROC curve can be used. This curve shows the performance of the model at all classification thresholds, by calculating a False Positive Rate (FPR) and a True Positive Rate (TPR) for each threshold. These rates are calculated with

$$\mathrm{FPR_i} = \frac{\mathrm{FP_i}}{\mathrm{FP_i} + \mathrm{TN_i}} \tag{3.1}$$

$$\mathrm{TPR_i} = \frac{\mathrm{TP_i}}{\mathrm{TP_i} + \mathrm{TN_i}} \tag{3.2}$$

Where FP is the number of False Positives, TN is the number of True negatives and TP is the number of True Positives for every threshold value i. In this study this is applied to the results of a regression model, to determine the optimal classification threshold. For every threshold, the experimental values are divided in Positives and Negatives, then using the predicted values, the values of FP, TN, TP and False Negatives (FN) are determined.

The TPR is then plotted against the FPR, using the corresponding threshold values. A model that performs poorly in classification will have a curve close to the TPR=FPR diagonal, while a model that performs well, will show TPR values close to 1. This can be quantified by calculating the area under the curve, which yields a performance metric that is referred to as AUC-ROC, Area Under Curve of Receiver Operating Characteristic.

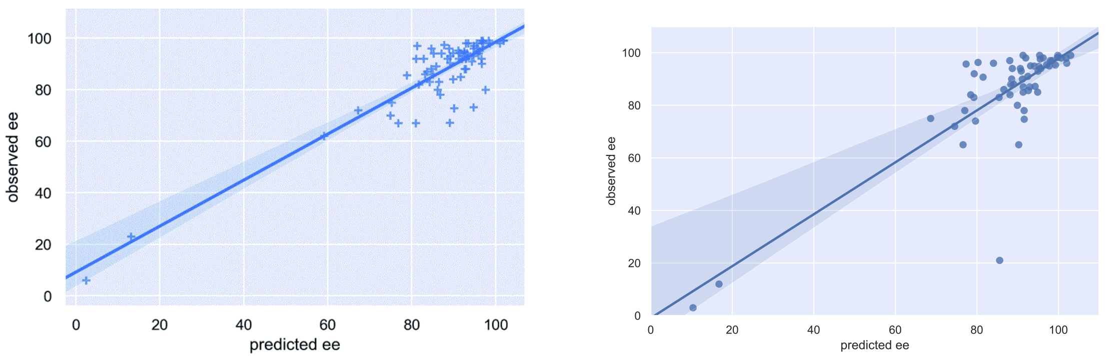
4. Results and discussion

4.1 Transfer learning on data of Singh and Sunoj

As explained in Section 1, the work of Singh and Sunoj [21] is the basis for this work. The first step towards application of Language models to our own data is to reproduce their findings. This will confirm the validity of using their workflow as a basis, give a good understanding of the structure of the workflow, and makes it easier to use their workflow on our own data.
Singh and Sunoj used Jupyter Notebooks with Google Colab, which are provided in a Github repository [46]. Here, the pretrained weights, the vocabulary for the pretrained model, and the data used for fine-tuning is also provided.
As mentioned, the experiment of Singh and Sunoj was done in Google Colab. In reproducing this, a few hurdles had to be taken:

- Singh and Sunoj performed their experiment with a deprecated version of the Python package FastAI v1 [44].
- This version relies on a deprecated functionality in an older version of the Python package Numpy ($\leq$1.21.6)
- A few errors in the code provided.

Since the API of FastAI v1 is not compatible with the API of FastAI v2, FastAI v1 had to be used to reproduce the experiments. This was installed, together with Numpy version 1.21.6, from which a deprecated array functionality was used. Furthermore, some incomplete code was completed, and some errors were fixed. Performing a single experiment, with seed [1234] yielded an RMSE value of 11.3. As mentioned in 2.3, Singh and Sunoj found an average RMSE of 8.38 over 30 experiments with different train-valid-test splits, with RMSE values ranging from 5.9 to 11.6.



(a) (b)

Figure 4.1: Comparison of prediction of ee values: (a) as reported by Singh and Sunoj [21], and (b) as found in our study, using the data Singh and Sunoj provided

Figure 4.1 show a comparison of the performance of the model as reported, and the performance of the model as found. In this comparison, two things have to be noticed. First, the dataset is quite skewed, with a large number of datapoints towards the higher end of the $[0, 100]$ interval for enantiomeric excess. Secondly, in the case of Singh and Sunoj, the model was able to predict the low enantiomeric excess quite well, despite the low number of datapoints in that region. In our case, there is a large outlier with an experimental ee of 21, which the model predicted quite high. Excluding this datapoint gives an RMSE of 7.5, which is in line with the findings of Singh and Sunoj.

These values were found without initially fine-tuning the encoder weights on the target data. Singh and Sunoj showed that this does not have an effect, or even acts negatively in terms of performance of the model. This was confirmed in an experiment, raising the RMSE to 8.5. While adverse effects are difficult to explain, similarity in the SMILES characters used for pre-training and target data can explain that no effect is visible.

An attempt was made to convert the notebook to Python script, to be able to run on supercomputer Snellius or DelftBlue cluster. This was however not directly possible, as FastAI v1 requires Numpy $\leq$1.21.6, which requires Python $\leq$3.10, which is not readily available on Snellius and DelftBlue. Using the documentation and courses of FastAI [47], the code was rewritten to use the API of FastAI v2, with a thorough look into the functionalities, to achieve similar model performance.

Major differences between the API's are mainly found in constructing the dataloaders. Furthermore, a change in the way padding is applied to batches requires the use of a higher API level in FastAI v2. Last, it is reported that FastAI v2 uses a newer version of the python package Torch, changing the way weights are saved in checkpoint files. Most importantly, a deep dive into the LSTM model used in both FastAI versions makes clear that the model architecture hyperparameters are kept the same between the versions. These correspond with the hyperparameters mentioned in Section 2.3.

This was extensively tested and explored, but the same performance could not be achieved. The performance of the model rewritten with FastAI version 2 gave a RMSE of 14.6, compared to an RMSE of 8.38 reported with Singh and Sunoj[21] and an RMSE of 11.3 found with FastAI v1.

For completeness, the workflow with LM-fine-tuning on the target data was also tested in FastAI v2. While it is expected that there is no improvement in performance, a change in the way the model is saved and loaded may change this behaviour. It was found, similarly to the findings reported, that no performance improvement was noticable (RMSE of 14.8).

4.2 Similarity of datasets

To investigate the applicability of the model of Sunoj on our own data, the similarity was investigated. First, a Principle Component Analysis was performed, to compare the ligands in both datasets.

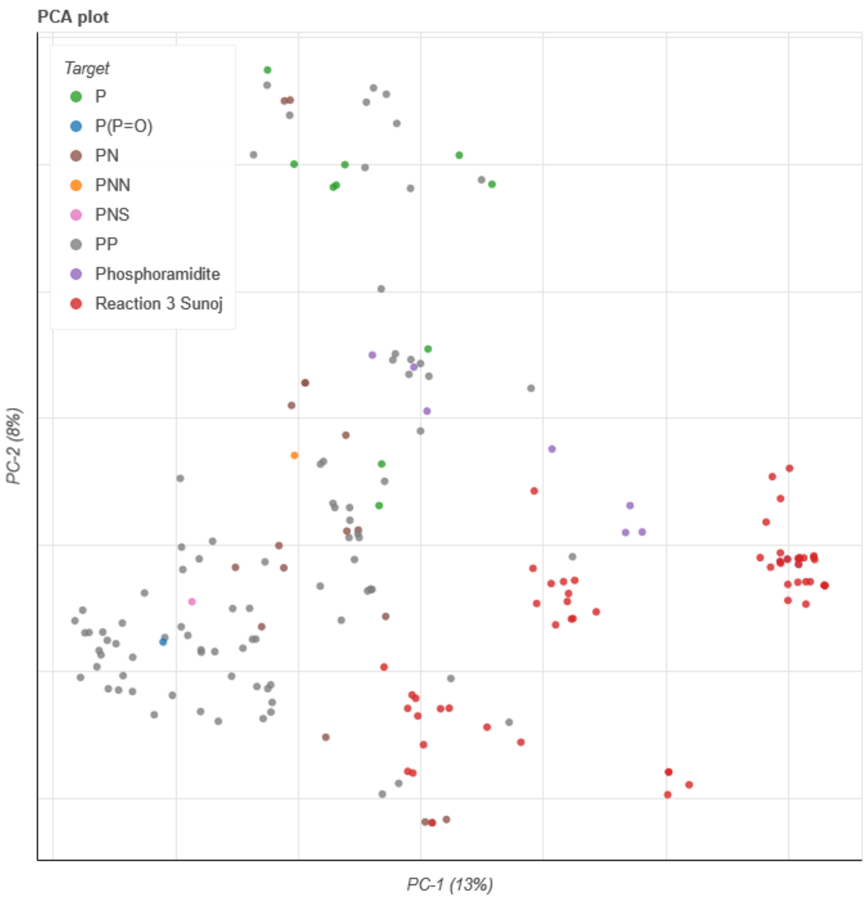


Figure 4.2: Principal Component Analysis to compare the chemical space of the ligands used in the study of Singh and Sunoj and in the data used in this work

Figure 4.2 shows the PCA-plot, with a reasonable good total explained variance of about 21%. In this plot, the clusters cannot be identified by ligand class (P, PP, PN, ...) or ligand family (BINAP, JosiPhos, ...). It is

however noticeable that the ligands of Singh and Sunoj form a much smaller cluster than the ligands in our data, pointing to a smaller spread in ligands used.
It is important to notice that the principal components are calculated from a large range of structural descriptors, calculated with the Python package Mordred. When inspecting the results of this descriptor calculation step, it was noticed that ferrocene-ligands gave an empty entry. Upon further investigation, it became clear that the structure of the ferrocene ligand results in SMILES data, where some carbon atoms have a valence of 5.

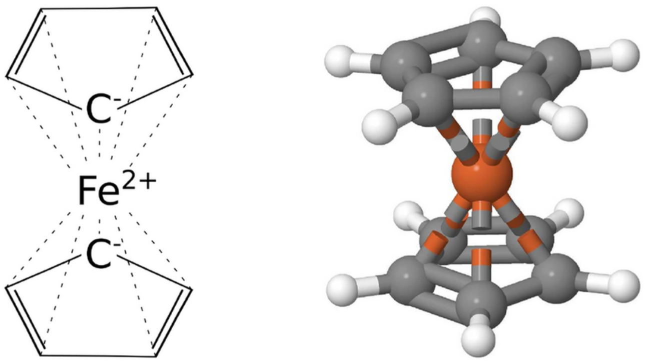


Figure 4.3: 2D and 3D structure of Ferrocene, showing the zero-order coordination bonds between the carbon atoms in the aromatic rings and the iron metal center. From [48]

Figure 4.3 shows the structure of ferrocene, showing the coordination bonds between the aromatic rings and the iron metal center. With this knowledge, and the fact that around 30% of the ligands in our dataset are ferrocenes, it can be assumed that the differences between the datasets is even larger than the PCA shows.

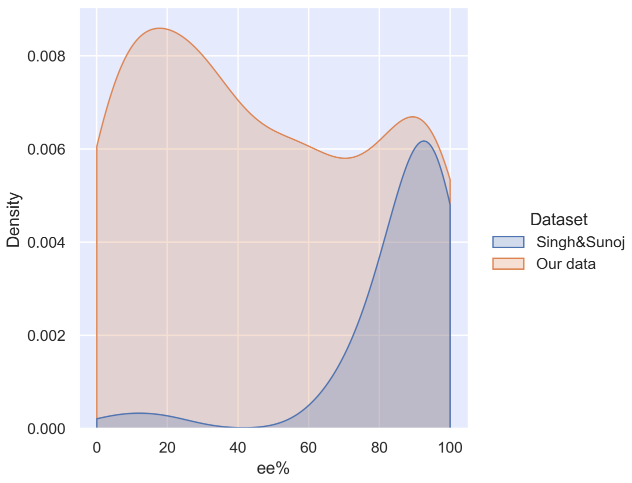


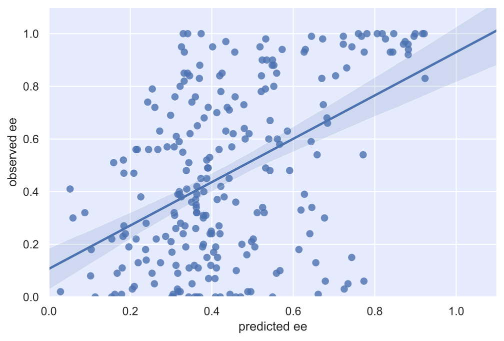
Figure 4.4: Comparison of distributions of $|ee|\%$ values in the dataset of Singh and Sunoj and our data

Investigations into the distribution of ee values in both datasets show other large differences. Figure 4.4 show a comparison of the distribution of both datasets. It is visible that the distribution of the ee% values of the dataset of Singh and Sunoj is quite skewed towards 100%, while the datapoints in our dataset is quite evenly distributed.

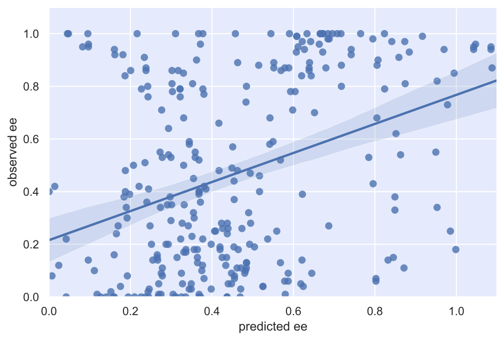
4.3 Transfer learning with own data

This analysis shows a sufficiently large difference between the datasets where the model Singh and Sunoj fine-tuned cannot be effectively used on our data. However, the pre-trained model can still be used, and fine-tuned to our own data and target task, using the MolPMoFiT approach [20], as explained in Section 2.2.3. This was done, as explained in Section 3. The differences in datasets required a small change in vocabulary. This will presumably have an impact on the performance, and will be checked by comparing the performance of the model with and without LM fine-tuning on target data.

Fine-tuning the model using the training hyperparameters as used by Singh and Sunoj [21], resulted in a relatively bad performance, as well as fine-tuning the model after fine-tuning the LM on the target data. Comparison of the results is shown in Figure 4.5, where fine-tuning only the regressor on the target task resulted in a RMSE of 0.303, and fine-tuning both the LM and the regressor on the target data resulted in a RMSE of 0.325.



(a)



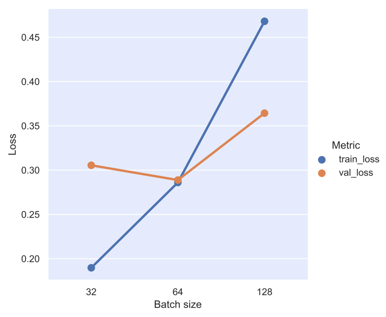
(b)

Figure 4.5: Comparison of prediction of ee values using the hyperparameters as given in Section 3: (a) fine-tuning of only regressor on target task, and (b) fine-tuning of LM and regressor on target task

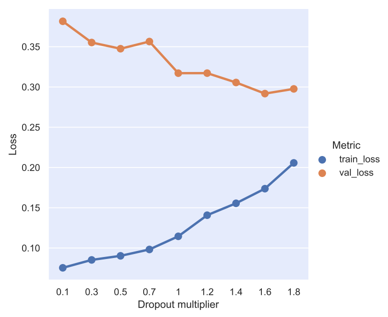
4.4 Hyperparameter optimization

To increase the performance of the model, a search for the optimal combination of hyperparameters was initiated. These parameters are, in order of investigation, batch size, dropout rate, number of epochs, learning rate, momentums and weight decay. The effect of these hyperparameters on the model performance was shown and explained in section 2.2.3. These hyperparameters can be optimized both for LM fine-tuning on the target data, and the regressor fine-tuning on the target task.

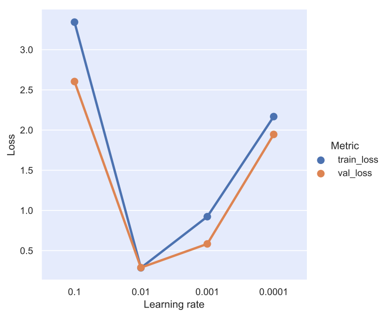
4.4.1 LM fine-tuning



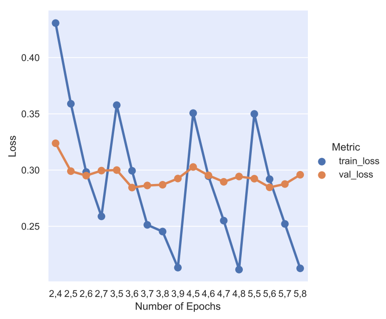
(a)



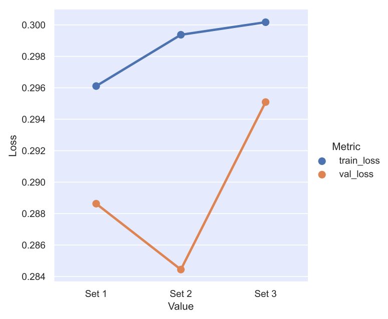
(b)



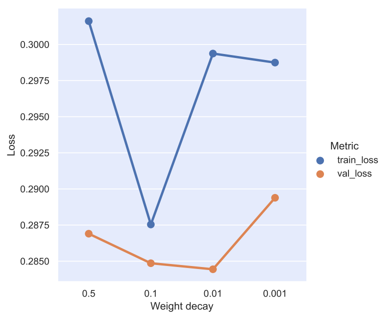
(c)



(d)



(e)



(f)

Figure 4.6: Overview of results of the fine-tuned hyperparameters for LM fine-tuning on target data. Some datapoints are labeled with 'Set #', the actual values are found in Appendix 5.1.

Figure 4.6 shows an overview of the performance of the Language Model learner, for a range of values for each hyperparameter that was analyzed. In each graph, the lowest point for validation loss is selected as optimal setting for the hyperparameter. The optimal selection of hyperparameters selected from this, for this case, is shown in Table 4.1. Note that the learning rate for the first cycle differs from the one shown in Figure 4.6c, since this is calculated for the learning rate in cycle 2 as:

$$\text{Learning rate cycle 1} = \text{Batch size}/48 * 10 * \text{Learning rate cycle 2} \tag{4.1}$$

Furthermore, it should be noticed that the dropout multiplier hyperparameter is not the dropout rate itself, but a multiplier of this value. Since the LSTM layers all have their own dropout rate by definition, this hyperparameter multiplies these dropout rates.

	Hyperparameter	Value
Learner hyperparameters		
	Dropout multiplier	1.6
	Batch size	64
LM training cycle 1		
Only embeddings		
	Number of epochs	3
	Learning rate	0.133
	Momentums	(0.8, 0.7, 0.8)
	Weight decay	0.01
LM training cycle 2		
All layers		
	Number of epochs	6
	Learning rate	0.01
	Momentums	(0.8, 0.7, 0.8)
	Weight decay	0.01

Table 4.1: Overview of tuned hyperparameters, and their optimal value found, for LM fine-tuning on target data.

4.4.2 Regressor fine-tuning

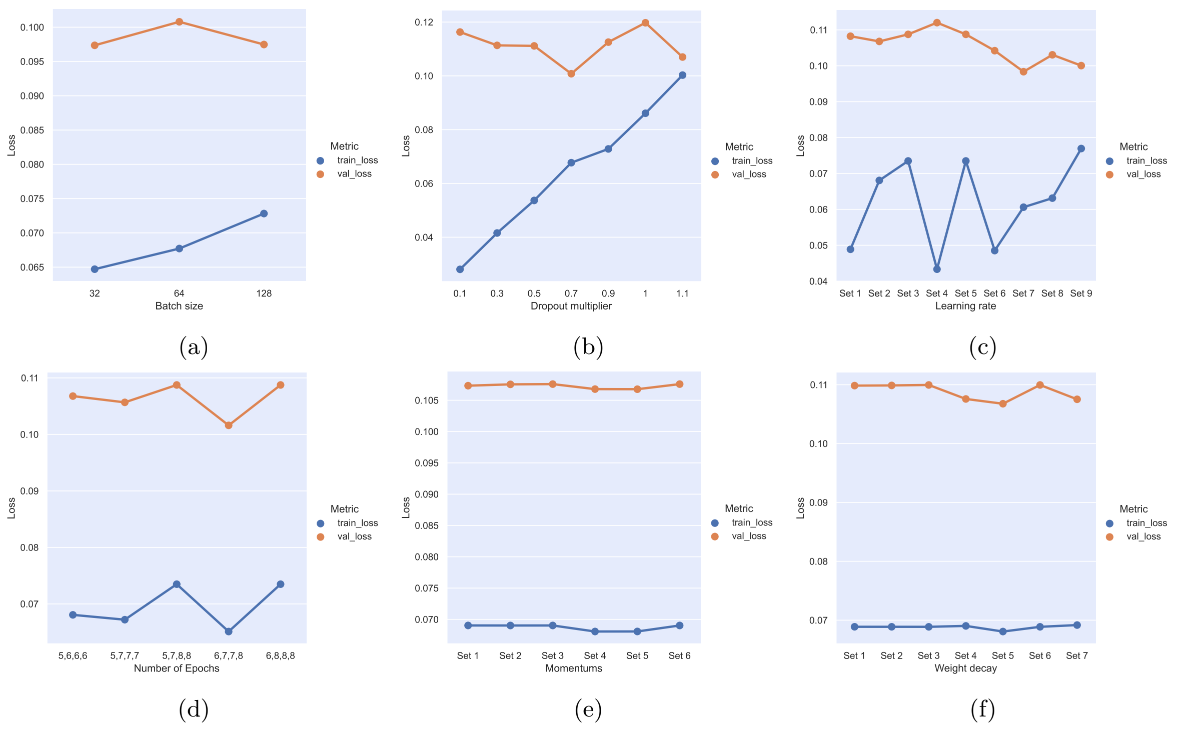
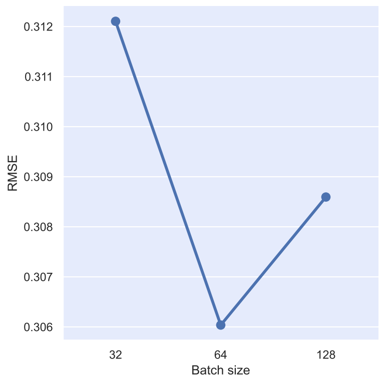
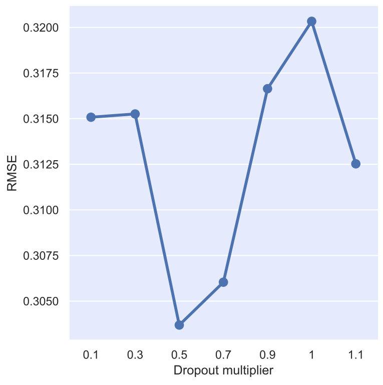


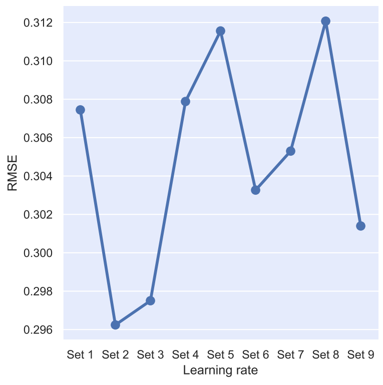
Figure 4.7: Overview of losses, as result of the fine-tuned hyperparameters for regressor fine-tuning on target task.Some datapoints are labeled with 'Set #', the actual values are found in Appendix 5.1.



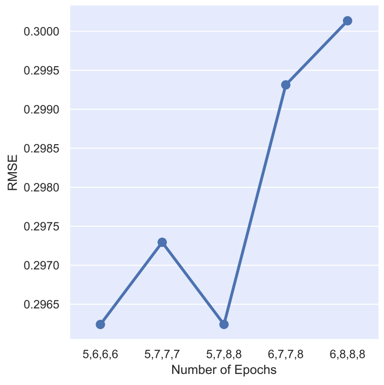
(a)



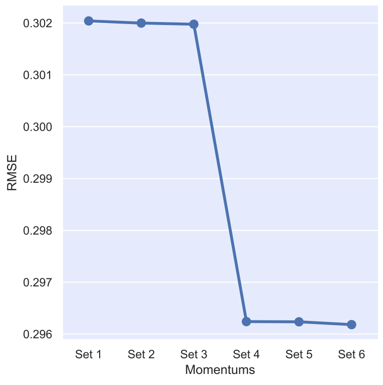
(b)



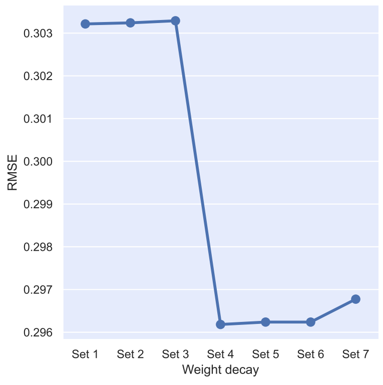
(c)



(d)



(e)



(f)

Figure 4.8: Overview of RMSE, as result of the fine-tuned hyperparameters for regressor fine-tuning on target task. Some datapoints are labeled with 'Set #', the actual values are found in Appendix 5.1.

Figures 4.7 and 4.8 shows an overview of the hyperparameters optimized, together with the performance metric, in this case RMSE. From this overview, it becomes clear that the optimal set of hyperparameters is as given in Table 4.2

	Hyperparameter	Value
Learner hyperparameters		
	Dropout multiplier	0.7
	Batch size	64
Training cycle 1		
Only regressor		
	Number of epochs	5
	Learning rate	0.1
	Momentums	(0.8, 0.7, 0.8)
	Weight decay	0.01
Training cycle 2		
Regressor and last encoder layer		
	Number of epochs	6
	Learning rate	0.001
	Momentums	(0.8, 0.7, 0.8)
	Weight decay	0.01
Training cycle 3		
Regressor and last two encoder layers		
	Number of epochs	6
	Learning rate	0.0001
	Momentums	(0.9, 0.8, 0.9)
	Weight decay	0.01
Training cycle 4		
Regressor and all encoder layers		
	Number of epochs	6
	Learning rate	0.0001
	Momentums	(0.9, 0.8, 0.9)
	Weight decay	0.01

Table 4.2: Overview of tuned hyperparameters, and their optimal value found, for fine-tuning of regressor on target task

Using these optimized hyperparameters, the performance is measured on the test set, resulting in a RMSE of 0.296. The results are shown in Figure 4.9, where it is visible that the model predicts most of the ee values between 0.3 and 0.7, while the true ee values are evenly distributed between 0 and 1. Furthermore, while there is quite a large number of good performing samples falsely predicted with a low ee value, only a few samples are wrongly predicted with a high ee value.

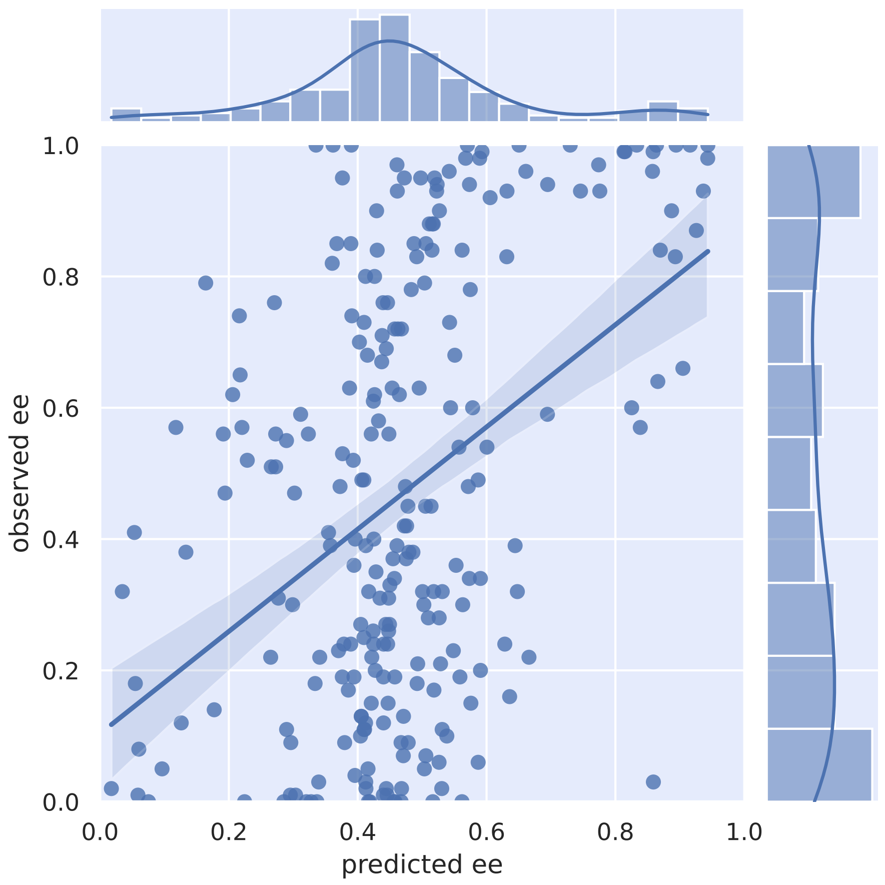
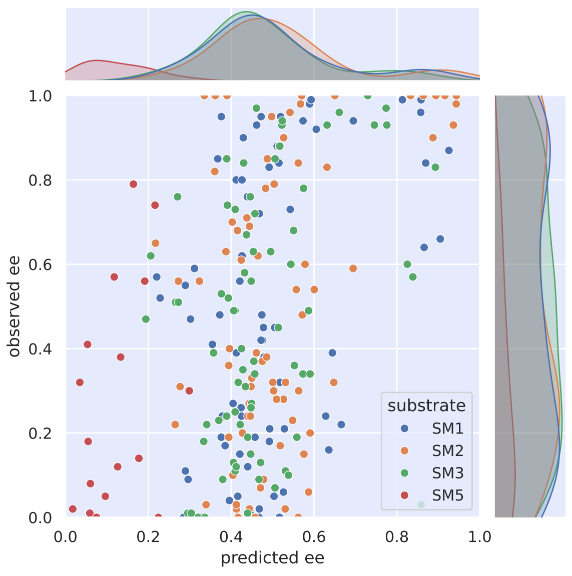
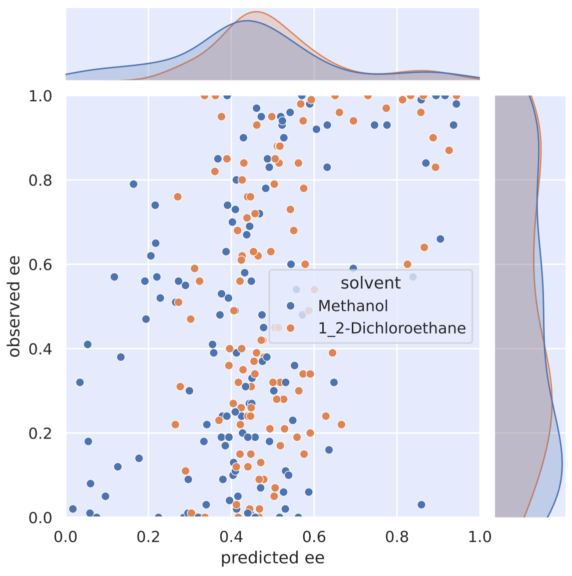


Figure 4.9: Prediction result on test set using optimized model

To investigate whether the selection of substrates, solvents and ligand family has an effect on the performance, these were investigated more closely. Figure 4.10a shows an overview of the performance of the model with the results separated per substrate, Figure 4.10b shows the performance of the model with the results separated per solvent. It is difficult to find a pattern in this, however, in the case of substrate 5, it is visible that the data is distributed towards the lower end of the range, which is reflected strongly in the predicted values. Furthermore, while not significantly different, it seems that the small difference in distribution between the two solvents is represented in the predicted ee value distribution.



(a)



(b)

Figure 4.10: Analysis of predicted $|ee|$ values, (a) by substrate, (b) by solvent

As mentioned in Section 2.2.2, one known limitation of RNN models is its difficulty to capture relations in long sequences. While there are no hard limits provided, sequences of 200 or more tokens are considered long for these kind of models [49]. In an attempt to show the influence of the sequence length on the accuracy of the prediction, these are plotted together in Figure 4.11. Unfortunately, aside from a few exceptions, no clear relation exists between sequence length and accuracy of prediction.

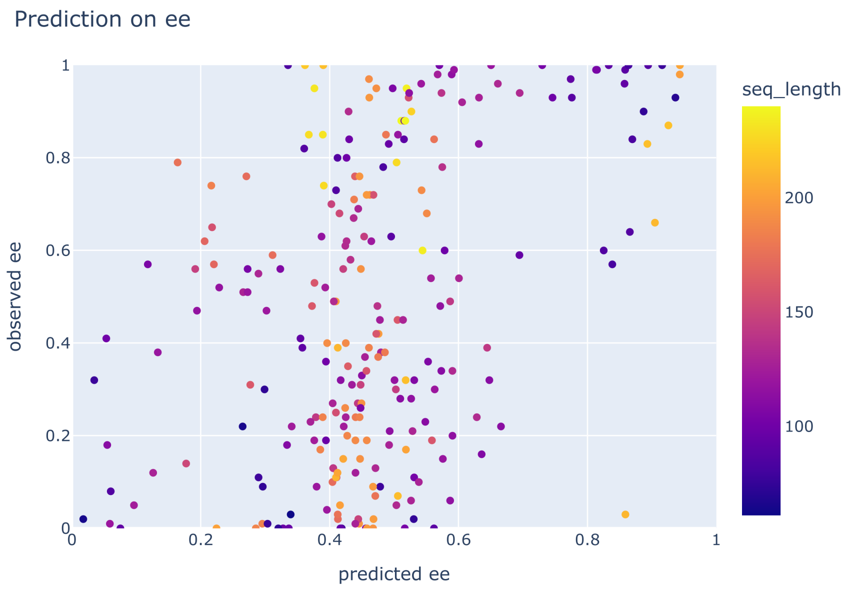


Figure 4.11: Analysis of effect of sequence length of performance of the model

These results show that the model is not accurately predicting ee values, and especially failing to predict values in the ee ranges below 0.3 and above 0.7. Since most values are close to average, and the goal of employing such a model is to accurately predict catalyst candidates, classifying the predicted $|ee|$ values can show the usefulness of the model.

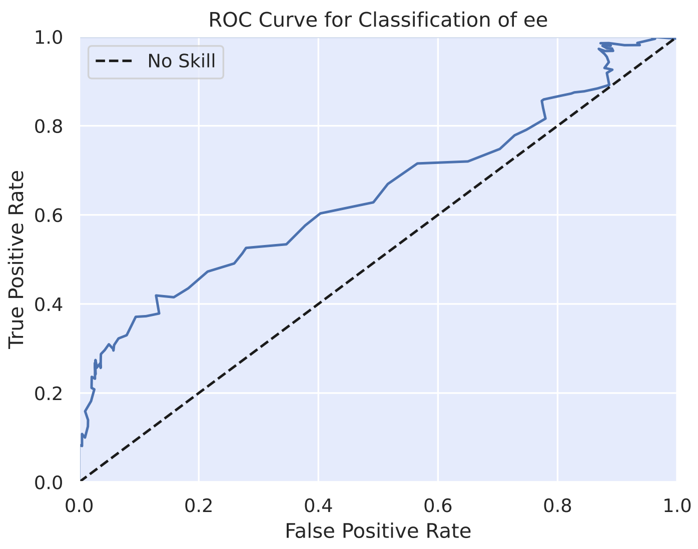
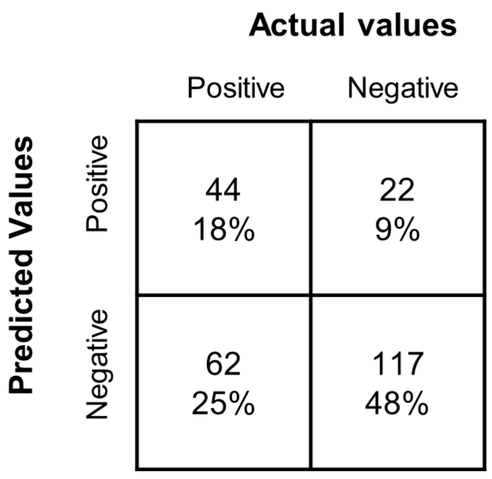


Figure 4.12: ROC curve of using model predictions on $|ee|$ in classification

Using ROC curve analysis, explained in Section 3, is was analyzed to what extent the model predictions can be used with classification, to provide catalyst candidates. The ROC curve is shown in Figure 4.12, which gives an optimal threshold of 0.53. Using this threshold, a confusion matrix can be created, shown in Figure 4.13



		Actual values	
		Positive	Negative
Predicted Values	Positive	44 18%	22 9%
	Negative	62 25%	117 48%

Figure 4.13: Confusion matrix of classification of model with threshold $|ee| = 0.53$

This allows to compare the performance of the model to experts in the field. After obtaining results of a classification with a selection of 28 datapoints, for each expert a confusion matrix can be created. Furthermore, the same procedure was used with a random chemist. The results of this comparison is shown in Figure 4.14, showing that the performance of the model created is comparable to experts in the field, especially indicating that a majority of the ligands result in a low ee.

		Actual values	
		Positive	Negative
Predicted Values	Positive	6 21%	5 17%
	Negative	5 17%	13 45%

(a) Expert 1

		Actual values	
		Positive	Negative
Predicted Values	Positive	6 21%	11 38%
	Negative	5 17%	7 24%

(b) Expert 2

		Actual values	
		Positive	Negative
Predicted Values	Positive	5 17%	5 17%
	Negative	5 17%	13 45%

(c) Expert 3

		Actual values	
		Positive	Negative
Predicted Values	Positive	5 17%	5 17%
	Negative	6 21%	13 45%

(d) Expert 4

		Actual values	
		Positive	Negative
Predicted Values	Positive	4 14%	11 38%
	Negative	7 24%	7 24%

(e) Random chemist

		Actual values	
		Positive	Negative
Predicted Values	Positive	44 18%	22 9%
	Negative	62 25%	117 48%

(f) NLP model

Figure 4.14: Overview of the results of expert classification of a selection of 28 datapoints. The actual results per expert are provided in Appendix 5.1.

5. Conclusion

Using the workflow used by Singh and Sunoj, an attempt was made to recreate a language model, and reproduce the results of their work using this model and the data provided. While the performance of the created model (RMSE = 9.9) could not match the reported performance (RMSE = 8.38), but almost reached the reported performance range (RMSE = 8.38 +/- 1.40). Since only one train-valid-test split was used in an experiment, no solid conclusions can be made about whether the data and information provided in the paper and supplementary information is not enough to reproduce the experiment.

Using the same workflow as a basis, the API of FastAI version 2 was used to reconstruct the model, using the data provided in the mentioned study. To use the same model, hyperparameters and a similar workflow, it was necessary to use a higher level of API, making it less straightforward in application than using FastAI version 1, in this context. The reported performance of the model could not be reproduced, as well as the experiment using FastAI version 1. The performance found using FastAI version 2 was with an RMSE of 14.6. This shows that it is not directly possible to transfer workflows or models from FastAI version 1 to version 2. Since no hyperparameter optimization was performed, it is not possible to conclude that the same performance could not be reached in FastAI version 2.

A Principal Component Analysis was performed on the datasets of ligands for assymetric hydrogenation used in the study of Singh and Sunoj and the ligands in a dataset for asymmetric hydrogenation from the ISE research group, TU Delft. While this analysis showed some similarity, it highlighted the larger spread in the ISE dataset. This analysis could not be performed for 30% of the dataset, due to notation of zero-order coordination bonds in SMILES.

The workflow developed by Li and Fourches, applied by Singh and Sunoj, and transferred to FastAI v2 in this study, was applied to the ISE dataset mentioned. Due to the notation issues mentioned above, data augmentation could not be used. With the workflow applied, a RMSE of 0.303 was achieved for $|ee|$ values in the range $[0, 1]$. This performance is notably worse from the performance the model showed on the dataset of Singh and Sunoj, this can be attributed to the larger spread in ligands, and the inability to use the data augmentation from the existing workflow.

The training hyperparameters in this workflow were optimized for performance on the ISE dataset. Both the hyperparameters of LM fine-tuning on the target data and regressor fine-tuning on the target task are optimized separately. The resulting model is analyzed for regression performance, which was poor, with a RMSE of 0.296. Furthermore, using ROC analysis, the performance of classification was examined. Considering the goal of suggesting ligand candidates for asymmetric hydrogenation, this model can reduce the amount of experiments required by around 73% , while classifying around 1 in 2 of good catalyst as bad candidates.

In short, the existing workflow could not directly be applied to a new, unseen dataset for enantioselective hydrogenation. Adjusting and optimizing the workflow yielded a model for predicting $|ee|$ of asymmetric hydrogenation ligands, performing rather poor in regression, and decent in classification.

5.1 Recommendations

To improve performance of the model used, some changes to the workflow can be made. First of all, one of the issues that was run into during the application of the existing workflow to the new dataset was the inability to use the data augmentation routine. This keeps the number of datapoints low, and can be a major contribution to the rather large regression error. To confirm this, increase in the train-valid ratio should show increase in performance. Data augmentation can then only be used if the representation of the ferrocene ligands in SMILES is changed.

Also, as Li and Fourches reported [29], using SMILES Pair Encoding (SPE) instead of atom-wise tokenization as tokenizer can improve performance as well. The same researchers show in other work that training the model as classifier can provide higher accuracy results as well [20].

Furthermore, using a transformer architecture instead of an AWD-LSTM can improve performance. Transformer models process a sequence at once, assigning attention weights to determine the most important parts of the sequence. This overcomes mostly the difficulty with processing longer sequences that RRN models show [33].

Last, Jablonka et al. showed good performance on prediction of molecular properties, using large language models, with a transformer architecture, but, trained on a selection of scientific sources. These models were fine-tuned on the language of the target data. Using this approach, although the models are larger and more computationally expensive to train, a potentially higher performance can be achieved.

Bibliography

[1] Roger A. Sheldon. “Metrics of Green Chemistry and Sustainability: Past, Present, and Future”. In: *ACS Sustainable Chemistry & Engineering* 6.1 (Jan. 2, 2018), pp. 32–48. ISSN: 2168-0485, 2168-0485. DOI: 10.1021/acssuschemeng.7b03505. URL: https://pubs.acs.org/doi/10.1021/acssuschemeng.7b03505 (visited on 04/23/2024).

[2] Roger A Sheldon. “Organic synthesis-past, present and future”. In: *Chemistry and Industry* 23 (1992), pp. 903–6.

[3] Stefan Koenig. *Scalable Green Chemistry: Case Studies from the Pharmaceutical Industry.* Aug. 31, 2013. ISBN: 978-981-4316-49-1.

[4] Roger A. Sheldon. “Catalysis: The Key to Waste Minimization”. In: *Journal of Chemical Technology & Biotechnology* 68.4 (1997). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/%28SICI%291097-4660%28199704%2968%3A4%3C381%3A%3AAID-JCTB620%3E3.0.CO%3B2-3, pp. 381–388. ISSN: 1097-4660. DOI: 10.1002/(SICI)1097-4660(199704)68:4<381::AID-JCTB620>3.0.CO;2-3. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291097-4660%28199704%2968%3A4%3C381%3A%3AAID-JCTB620%3E3.0.CO%3B2-3 (visited on 06/12/2024).

[5] Piet W. N. M. van Leeuwen. *Homogeneous catalysis: understanding the art.* Dordrecht: Kluwer Acad. Publ, 2004. 407 pp. ISBN: 978-1-4020-3176-2.

[6] Carl A. Busacca et al. “The Growing Impact of Catalysis in the Pharmaceutical Industry”. In: *Advanced Synthesis & Catalysis* 353.11 (2011). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/adsc.201100488, pp. 1825–1864. ISSN: 1615-4169. DOI: 10.1002/adsc.201100488. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/adsc.201100488 (visited on 06/21/2024).

[7] Aleksei N. Marianov et al. “Homogeneous and heterogeneous strategies of enantioselective hydrogenation: Critical evaluation and future prospects”. In: *Chem Catalysis* 3.7 (July 20, 2023). Publisher: Elsevier. ISSN: 2667-1107, 2667-1093. DOI: 10.1016/j.checat.2023.100631. URL: https://www.cell.com/chem-catalysis/abstract/S2667-1093(23)00152-5 (visited on 04/22/2024).

[8] Katrin Rosenthal and Stephan Lütz. “Recent developments and challenges of biocatalytic processes in the pharmaceutical industry”. In: *Current Opinion in Green and Sustainable Chemistry.* Pharmaceuticals / Green Processes and Technologies 11 (June 1, 2018), pp. 58–64. ISSN: 2452-2236. DOI: 10.1016/j.cogsc.2018.03.015. URL: https://www.sciencedirect.com/science/article/pii/S2452223617300949 (visited on 06/12/2024).

[9] Adarsh V Kalikadien et al. “Optimizing Chiral Catalysts through Predictive Modelling and High-Throughput Experimentation Techniques”. In: ().

[10] Adarsh V. Kalikadien et al. “Paving the road towards automated homogeneous catalyst design”. In: *ChemPlusChem* n/a (n/a). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cplu.202300702, e202300702. ISSN: 2192-6506. DOI: 10.1002/cplu.202300702. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cplu.202300702 (visited on 02/26/2024).

[11] Andrew F. Zahrt et al. “Prediction of higher-selectivity catalysts by computer-driven workflow and machine learning”. In: *Science (New York, N.Y.)* 363.6424 (Jan. 18, 2019), eaau5631. ISSN: 0036-8075. DOI: 10.1126/science.aau5631. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6417887/ (visited on 06/12/2024).

[12] Kevin Maik Jablonka et al. “Leveraging large language models for predictive chemistry”. In: *Nature Machine Intelligence* 6.2 (Feb. 2024). Number: 2 Publisher: Nature Publishing Group, pp. 161–169. ISSN: 2522-5839. DOI: 10.1038/s42256-023-00788-1. URL: https://www.nature.com/articles/s42256-023-00788-1 (visited on 02/26/2024).

[13] Andrew D. White et al. “Assessment of chemistry knowledge in large language models that generate code”. In: *Digital Discovery* 2.2 (Apr. 11, 2023). Publisher: RSC, pp. 368–376. ISSN: 2635-098X. DOI: 10.1039/D2DD00087C. URL: https://pubs.rsc.org/en/content/articlelanding/2023/dd/d2dd00087c (visited on 06/20/2024).

[14] Derek T Ahneman et al. “Predicting reaction performance in C–N cross-coupling using machine learning”. In: *Science* 360.6385 (2018). Publisher: American Association for the Advancement of Science, pp. 186–190.

[15] Damith Perera et al. “A platform for automated nanomole-scale reaction screening and micromole-scale synthesis in flow”. In: *Science* 359.6374 (2018). Publisher: American Association for the Advancement of Science, pp. 429–434.

[16] Yasuhiro Yoshikai et al. "Difficulty in chirality recognition for Transformer architectures learning chemical structures from string representations". In: *Nature Communications* 15.1 (Feb. 16, 2024). Publisher: Nature Publishing Group, p. 1197. ISSN: 2041-1723. DOI: 10.1038/s41467-024-45102-8. URL: https://www.nature.com/articles/s41467-024-45102-8 (visited on 03/18/2024).

[17] Ines Filipa Martins et al. "A Bayesian approach to in silico blood-brain barrier penetration modeling". In: *Journal of chemical information and modeling* 52.6 (2012). Publisher: ACS Publications, pp. 1686–1697.

[18] David L. Mobley and J. Peter Guthrie. "FreeSolv: a database of experimental and calculated hydration free energies, with input files". In: *Journal of Computer-Aided Molecular Design* 28.7 (July 2014), pp. 711–720. ISSN: 0920-654X, 1573-4951. DOI: 10.1007/s10822-014-9747-x. URL: http://link.springer.com/10.1007/s10822-014-9747-x (visited on 06/21/2024).

[19] Jian-Bing Wang et al. "*In silico* evaluation of \textlogD_7.4 and comparison with other prediction methods". In: *Journal of Chemometrics* 29.7 (2015). Publisher: Wiley Online Library, pp. 389–398.

[20] Xinhao Li and Denis Fourches. "Inductive transfer learning for molecular activity prediction: Next-Gen QSAR Models with MolPMoFiT". In: *Journal of Cheminformatics* 12.1 (Apr. 22, 2020), p. 27. ISSN: 1758-2946. DOI: 10.1186/s13321-020-00430-x. URL: https://doi.org/10.1186/s13321-020-00430-x (visited on 03/12/2024).

[21] Sukriti Singh and Raghavan B. Sunoj. "A transfer learning protocol for chemical catalysis using a recurrent neural network adapted from natural language processing". In: *Digital Discovery* 1.3 (June 13, 2022). Publisher: RSC, pp. 303–312. ISSN: 2635-098X. DOI: 10.1039/D1DD00052G. URL: https://pubs.rsc.org/en/content/articlelanding/2022/dd/d1dd00052g (visited on 02/26/2024).

[22] Stanisław Wacławek, Vinod V. T. Padil, and Miroslav Černík. "Major Advances and Challenges in Heterogeneous Catalysis for Environmental Applications: A Review". In: *Ecological Chemistry and Engineering S* 25.1 (Mar. 1, 2018), pp. 9–34. URL: https://sciendo.com/article/10.1515/eces-2018-0001 (visited on 03/08/2024).

[23] W. Knowles, M. Sabacky, and B. Vineyard. "CATALYTIC ASYMMETRIC HYDROGENATION USING SOLUBLE, OPTICALLY ACTIVE PHOSPHINE COMPLEXES". In: *Annals of the New York Academy of Sciences* 172 (Dec. 2006), pp. 232–237. DOI: 10.1111/j.1749-6632.1970.tb34979.x.

[24] L. Horner, H. Siegel, and H. Büthe. "Asymmetric Catalytic Hydrogenation with an Optically Active Phosphinerhodium Complex in Homogeneous Solution". In: *Angewandte Chemie International Edition in English* 7.12 (1968). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/anie.196809422, pp. 942–942. ISSN: 1521-3773. DOI: 10.1002/anie.196809422. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/anie.196809422 (visited on 06/13/2024).

[25] Elemer Fogassy et al. "Optical resolution methods". In: *Organic & Biomolecular Chemistry* 4.16 (2006). Publisher: Royal Society of Chemistry, pp. 3011–3030.

[26] Chris SG Seo and Robert H Morris. "Catalytic homogeneous asymmetric hydrogenation: successes and opportunities". In: *Organometallics* 38.1 (2018). Publisher: ACS Publications, pp. 47–65.

[27] Rico Sennrich, Barry Haddow, and Alexandra Birch. *Neural Machine Translation of Rare Words with Subword Units*. June 10, 2016. DOI: 10.48550/arXiv.1508.07909. arXiv: 1508.07909[cs]. URL: http://arxiv.org/abs/1508.07909 (visited on 06/18/2024).

[28] Taku Kudo and John Richardson. *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing*. Aug. 19, 2018. DOI: 10.48550/arXiv.1808.06226. arXiv: 1808.06226[cs]. URL: http://arxiv.org/abs/1808.06226 (visited on 06/18/2024).

[29] Xinhao Li and Denis Fourches. "SMILES Pair Encoding: A Data-Driven Substructure Tokenization Algorithm for Deep Learning". In: *Journal of Chemical Information and Modeling* 61.4 (Apr. 26, 2021). Publisher: American Chemical Society, pp. 1560–1569. ISSN: 1549-9596. DOI: 10.1021/acs.jcim.0c01127. URL: https://doi.org/10.1021/acs.jcim.0c01127 (visited on 03/08/2024).

[30] Jatin Karthik Tripathy et al. "Comprehensive analysis of embeddings and pre-training in NLP". In: *Computer Science Review* 42 (Nov. 1, 2021), p. 100433. ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2021.100433. URL: https://www.sciencedirect.com/science/article/pii/S1574013721000733 (visited on 06/18/2024).

[31] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems*. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html (visited on 06/18/2024).

[32] AWS. *What is RNN? - Recurrent Neural Networks Explained.* Amazon Web Services, Inc. URL: https://aws.amazon.com/what-is/recurrent-neural-network/ (visited on 06/19/2024).

[33] Ashish Vaswani et al. "Attention is All you Need". In: ().

[34] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. *Regularizing and Optimizing LSTM Language Models.* Aug. 7, 2017. DOI: 10.48550/arXiv.1708.02182. arXiv: 1708.02182[cs]. URL: http://arxiv.org/abs/1708.02182 (visited on 06/18/2024).

[35] Aysu Ezen-Can. *A Comparison of LSTM and BERT for Small Corpus.* Sept. 11, 2020. DOI: 10.48550/arXiv.2009.05451. arXiv: 2009.05451[cs]. URL: http://arxiv.org/abs/2009.05451 (visited on 06/20/2024).

[36] Sinno Jialin Pan and Qiang Yang. "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (Oct. 2010). Conference Name: IEEE Transactions on Knowledge and Data Engineering, pp. 1345–1359. ISSN: 1558-2191. DOI: 10.1109/TKDE.2009.191. URL: https://ieeexplore.ieee.org/abstract/document/5288526 (visited on 06/20/2024).

[37] Jeremy Howard and Sebastian Ruder. *Universal Language Model Fine-tuning for Text Classification.* May 23, 2018. DOI: 10.48550/arXiv.1801.06146. arXiv: 1801.06146[cs,stat]. URL: http://arxiv.org/abs/1801.06146 (visited on 06/18/2024).

[38] Jason Yosinski et al. *How transferable are features in deep neural networks?* _eprint: 1411.1792. 2014.

[39] Lizhi Liao et al. "An Empirical Study of the Impact of Hyperparameter Tuning and Model Optimization on the Performance Properties of Deep Neural Networks". In: *ACM Transactions on Software Engineering and Methodology* 31.3 (July 31, 2022), pp. 1–40. ISSN: 1049-331X, 1557-7392. DOI: 10.1145/3506695. URL: https://dl.acm.org/doi/10.1145/3506695 (visited on 06/20/2024).

[40] John Burgess. "RTX on—The NVIDIA Turing GPU". In: *IEEE Micro* 40.2 (Mar. 2020). Conference Name: IEEE Micro, pp. 36–44. ISSN: 1937-4143. DOI: 10.1109/MM.2020.2971677. URL: https://ieeexplore.ieee.org/document/8981896 (visited on 06/17/2024).

[41] *Snellius: the National Supercomputer | SURF.nl.* URL: https://www.surf.nl/en/services/snellius-the-national-supercomputer (visited on 06/17/2024).

[42] *NVIDIA A100 GPUs Power the Modern Data Center.* NVIDIA. URL: https://www.nvidia.com/en-us/data-center/a100/ (visited on 06/17/2024).

[43] Delft High Performance Computing Centre (DHPC). *DelftBlue Supercomputer (Phase 2).* 2024. URL: https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2.

[44] Jeremy Howard and Sylvain Gugger. "Fastai: A Layered API for Deep Learning". In: *Information* 11.2 (Feb. 16, 2020), p. 108. ISSN: 2078-2489. DOI: 10.3390/info11020108. URL: https://www.mdpi.com/2078-2489/11/2/108 (visited on 06/17/2024).

[45] Sunghwan Kim et al. "PubChem Compound Summary for CID 11, 1,2-Dichloroethane". In: *Nucleic Acids Research* 51 (D1 Jan. 6, 2023), pp. D1373–D1380. ISSN: 0305-1048, 1362-4962. DOI: 10.1093/nar/gkac956. URL: https://academic.oup.com/nar/article/51/D1/D1373/6777787 (visited on 06/14/2024).

[46] *Transfer_Learning_in_Catalysis.* June 2024. URL: https://github.com/Sunojlab/Transfer_Learning_in_Catalysis/tree/main (visited on 06/17/2024).

[47] J. Howard and S. Gugger. *Deep Learning for Coders with Fastai and Pytorch: AI Applications Without a PhD.* O'Reilly Media, Incorporated, 2020. ISBN: 978-1-4920-4552-6. URL: https://books.google.no/books?id=xd6LxgEACAAJ.

[48] Antanas Vaitkus et al. "A workflow for deriving chemical entities from crystallographic data and its application to the Crystallography Open Database". In: *Journal of Cheminformatics* 15 (Dec. 2023). DOI: 10.1186/s13321-023-00780-2.

[49] Greg Van Houdt, Carlos Mosquera, and Gonzalo Nápoles. "A review on the long short-term memory model". In: *Artificial Intelligence Review* 53.8 (Dec. 2020), pp. 5929–5955. ISSN: 0269-2821, 1573-7462. DOI: 10.1007/s10462-020-09838-1. URL: https://link.springer.com/10.1007/s10462-020-09838-1 (visited on 06/20/2024).

Appendix 1

Hyperparameter name	Value[1]	train_loss	val_loss
Batch size	32	0.189725	0.305471
Batch size	64	0.286071	0.288813
Batch size	128	0.468063	0.364342
Dropout multiplier	0.1	0.075425	0.381592
Dropout multiplier	0.3	0.085225	0.355239
Dropout multiplier	0.5	0.090293	0.347536
Dropout multiplier	0.7	0.098216	0.356452
Dropout multiplier	1	0.114524	0.317155
Dropout multiplier	1.2	0.14076	0.317203
Dropout multiplier	1.4	0.155558	0.30558
Dropout multiplier	1.6	0.17371	0.291784
Dropout multiplier	1.8	0.205731	0.297644
Learning rate	0.1	3.342852	2.603955
Learning rate	0.01	0.286071	0.288813
Learning rate	0.001	0.922876	0.583896
Learning rate	0.0001	2.167748	1.945783
Number of Epochs	2,4	0.430719	0.323842
Number of Epochs	2,5	0.358953	0.299027
Number of Epochs	2,6	0.298263	0.29506
Number of Epochs	2,7	0.258811	0.299512
Number of Epochs	3,5	0.357743	0.299983
Number of Epochs	3,6	0.299372	0.284442
Number of Epochs	3,7	0.251257	0.286306
Number of Epochs	3,8	0.245333	0.28693
Number of Epochs	3,9	0.213236	0.292478
Number of Epochs	4,5	0.350666	0.302734
Number of Epochs	4,6	0.294535	0.295255
Number of Epochs	4,7	0.25502	0.289546
Number of Epochs	4,8	0.211498	0.294283
Number of Epochs	5,5	0.349945	0.292332
Number of Epochs	5,6	0.291911	0.284634
Number of Epochs	5,7	0.252113	0.28757
Number of Epochs	5,8	0.212612	0.295824
Momentums	(0.7, 0.6, 0.7),(0.8, 0.7, 0.8)	0.296109	0.288631
Momentums	(0.8, 0.7, 0.8),(0.8, 0.7, 0.8)	0.299372	0.284442
Momentums	(0.8, 0.7, 0.8),(0.9, 0.8, 0.9)	0.30017	0.295092
Weight decay	0.5	0.301616	0.286904
Weight decay	0.1	0.287541	0.28486
Weight decay	0.01	0.299372	0.284442
Weight decay	0.001	0.298746	0.289389

Table 1: Overview of results of hyperparameter optimization of Language Learner. [1] When formatted as list, the values are corresponding to the respective learning cycle

Appendix 2

Hyperparameter name	Value[1]	train_loss	val_loss	RMSE	MAE	R2_score
Batch size	32	0.064703	0.097358	0.312105	0.267171	-1.35419
Batch size	64	0.067714	0.100791	0.306037	0.257568	-1.24459
Batch size	128	0.072818	0.097475	0.308595	0.264988	-1.91736
Dropout multiplier	0.1	0.02806	0.11629	0.31508	0.262126	-0.7584
Dropout multiplier	0.3	0.041573	0.111338	0.315258	0.263812	-0.81055
Dropout multiplier	0.5	0.05368	0.111138	0.303687	0.256021	-1.04005
Dropout multiplier	0.7	0.067714	0.100791	0.306037	0.257568	-1.24459
Dropout multiplier	0.9	0.072832	0.112538	0.316642	0.264249	-1.07883
Dropout multiplier	1	0.086097	0.119738	0.320336	0.267704	-1.67945
Dropout multiplier	1.1	0.100276	0.106993	0.312524	0.267519	-2.68835
Learning rate	[0.1, 0.01, 0.001, 0.001]	0.048877	0.108235	0.307446	0.255584	-0.77898
Learning rate	[0.1, 0.001, 0.0001, 0.0001]	0.068064	0.106782	0.296241	0.255142	-1.45504
Learning rate	[0.1, 0.0001, 1e-05, 1e-05]	0.073492	0.108751	0.297502	0.255433	-1.58614
Learning rate	[0.01, 0.01, 0.01, 0.001]	0.04333	0.112012	0.307883	0.251569	-0.64566
Learning rate	[0.01, 0.01, 0.001, 0.001]	0.073492	0.108751	0.311559	0.258336	-0.89421
Learning rate	[0.01, 0.01, 1e-05, 1e-05]	0.048519	0.104207	0.303265	0.254438	-1.00864
Learning rate	[0.01, 0.001, 1e-05, 1e-05]	0.060607	0.098346	0.305298	0.26109	-1.35625
Learning rate	[0.001, 0.001, 0.0001, 0.0001]	0.063132	0.103053	0.312069	0.266244	-1.60425
Learning rate	[0.001, 0.0001, 1e-05, 1e-05]	0.076946	0.100044	0.301396	0.262551	-2.59126
Number of Epochs	5,6,6,6	0.068064	0.106782	0.296241	0.255142	-1.45504
Number of Epochs	5,7,7,7	0.067208	0.105672	0.297294	0.256445	-1.49618
Number of Epochs	5,7,8,8	0.073492	0.108751	0.296241	0.255756	-1.44985
Number of Epochs	6,7,7,8	0.065111	0.1016	0.299312	0.256455	-1.44522
Number of Epochs	6,8,8,8	0.073492	0.108751	0.300134	0.257501	-1.46632
Momentums	[(0.7, 0.6, 0.7), (0.7, 0.6, 0.7), (0.8, 0.7, 0.8), (0.8, 0.7, 0.8)]	0.06903	0.107328	0.30204	0.260506	-1.53707
Momentums	[(0.7, 0.6, 0.7), (0.8, 0.7, 0.8), (0.8, 0.7, 0.8), (0.8, 0.7, 0.8)]	0.069023	0.107542	0.301999	0.260454	-1.53278
Momentums	[(0.7, 0.6, 0.7), (0.8, 0.7, 0.8), (0.9, 0.8, 0.9), (0.9, 0.8, 0.9)]	0.069027	0.107581	0.301977	0.260425	-1.5327
Momentums	[(0.8, 0.7, 0.8), (0.8, 0.7, 0.8), (0.8, 0.7, 0.8), (0.8, 0.7, 0.8)]	0.068064	0.106782	0.296241	0.255142	-1.45504
Momentums	[(0.8, 0.7, 0.8), (0.8, 0.7, 0.8), (0.9, 0.8, 0.9), (0.9, 0.8, 0.9)]	0.06807	0.106767	0.296236	0.255134	-1.45544
Momentums	[(0.8, 0.7, 0.8), (0.9, 0.8, 0.9), (0.9, 0.8, 0.9), (0.9, 0.8, 0.9)]	0.069027	0.107581	0.296182	0.255063	-1.44871
Weight decay	[0.1, 0.1, 0.1, 0.1]	0.068878	0.10985	0.303215	0.261998	-1.70917
Weight decay	[0.1, 0.1, 0.01, 0.01]	0.068875	0.109881	0.30324	0.262011	-1.70734
Weight decay	[0.1, 0.01, 0.01, 0.01]	0.068871	0.10996	0.303289	0.262001	-1.70055
Weight decay	[0.01, 0.01, 0.01, 0.01]	0.069027	0.107581	0.296182	0.255063	-1.44871
Weight decay	[0.01, 0.01, 0.001, 0.001]	0.06807	0.10677	0.296238	0.255135	-1.45528
Weight decay	[0.01, 0.001, 0.001, 0.001]	0.068871	0.10996	0.296238	0.255131	-1.45461
Weight decay	[0.001, 0.001, 0.001, 0.001]	0.06917	0.107524	0.296774	0.256554	-1.55481

Table 2: Overview of results of hyperparameter optimization of regression Learner. [1] When formatted as list, the values are corresponding to the respective learning cycle

Appendix 3

Ligand	Substrate	Sovent	Expert 1	Expert 2	Expert 3	Expert 4	Random Chemist
L3	SM1	Methanol	Yes	No	No	No	No
L22	SM1	Methanol	No	Yes	Yes	Yes	Yes
L38	SM1	Methanol	Yes	Yes	Yes	Yes	No
L72	SM1	Methanol	Yes	No	No	No	Yes
L114	SM1	Methanol	Yes	No	Yes	No	Yes
L15	SM1	1,2-Dichloroethane	Yes	No	No	No	Yes
L48	SM1	1,2-Dichloroethane	Yes	Yes	No	Yes	Yes
L192	SM1	1,2-Dichloroethane	No	Yes	No	No	Yes
L19	SM2	Methanol	No	Yes	Yes	Yes	Yes
L92	SM2	Methanol	No	Yes	No	No	No
L125	SM2	Methanol	No	Yes	No	No	No
L151	SM2	Methanol	No	Yes	No	No	No
L183	SM2	Methanol	No	No	Yes	No	Yes
L21	SM2	1,2-Dichloroethane	No	Yes	Yes	Yes	No
L137	SM2	1,2-Dichloroethane	No	Yes	No	Yes	Yes
L183	SM2	1,2-Dichloroethane	No	No	No	No	Yes
L22	SM3	Methanol	No	Yes	Yes	Yes	No
L48	SM3	Methanol	Yes	Yes	No	Yes	Yes
L108	SM3	Methanol	No	No	No	No	Yes
L125	SM3	Methanol	Yes	Yes	No	No	Yes
L38	SM3	1,2-Dichloroethane	Yes	Yes	Yes	Yes	no
L92	SM3	1,2-Dichloroethane	Yes	Yes	No	No	No
L115	SM3	1,2-Dichloroethane	Yes	No	Yes	No	No
L143	SM3	1,2-Dichloroethane	No	No	No	No	No
L108	SM5	Methanol	No	No	No	No	Yes
L114	SM5	Methanol	No	No	Yes	No	No
L122	SM5	Methanol	No	Yes	No	No	Yes
L143	SM5	Methanol	No	No	No	No	no
L192	SM5	Methanol	No	Yes	No	Yes	no

Table 3: Results of expert classification of enantionselectivity of selected ligands