The model is implemented using PyTorch deep learning framework and fast.ai library. All the calculations are run using the Google Colab Pro. The fast.ai library uses pre-trained language models, performs fine-tuning, in the following three steps:

(1) Data pre-processing.

(2) Use pre-trained weights to create a language model (LM) that can be used to fine-tune on the given dataset.

(3) Create models such as classifiers and/or regressors using the pre-trained or fine-tuned LM.

**Different steps involved in using Fastai’s ULMFiT**

**Step 1: Data Preprocessing**

First, a “databunch” is created for the given task that can be language modelling or classification/regression. The fast.ai provides simple functions to create the respective databunch and automatically performs the pre-processing of text.

Following two types of databunch can be created.

(1) TextLMDataBunch: it creates a databunch for the task of language modelling, and doesn’t require separate data labels. The data is processed in a manner such that the model learns to predict the next word given the previous words in a sentence.

(2) TextClasDataBunch: it creates a databunch for the task of classification/regression, and requires labelled data.

The fast.ai databunch can take the input in the form of dataframes. The train and validation (or test) dataframes has to be provided to build the databunch. The columns containing text and labels have to be specified. For building a databunch for language model, the label column can be ignored. The two main tasks when preparing the text data for modelling are: (a) tokenization, which splits the text into tokens, and (b) numericalization, that assigns a unique integer to each token. Either the fast.ai’s default tokenizer and numericalizer can be used or a custom tokenizer with specific vocabulary for numericalization can be passed. Once the pre-processing is done, the databunch can be saved. It can then be loaded as and when required.

**Step 2: Creating Language Model**

**Language model fine-tuning:**

This is the first step in training, where the pre-trained LM weights are used for fine-tuning on target data. Fast.ai provides an easy way of creating a “learner” (language\_model\_learner) for language model training. It requires LM databunch and a pre-trained model as input. AWD-LSTM is the model architecture used to train LM. There are other architectures available as well (e.g. transformer decoder, transformerXL). The ‘config’ argument can be used to customize the architecture. The ‘drop\_mult’ hyperparameter, can be tuned to set the amount of dropout that is a technique used for regularization. The pre-trained weights and the corresponding vocabulary can be passed to ‘pretrained\_fnames’ argument.

**Training the model:**

Learning rate (lr) is one of the most important hypeparameter in the training of a model. The fast.ai’s utility ‘lr\_find’ can be used to search a range of lr and the plot of lr versus loss is used to identify the lowest loss and choose the lr one magnitude higher than that corresponds to lowest point. The LM model then can be trained with this lr using ‘fit\_one\_cycle’. The fit\_one\_cycle takes the lr and number of epochs as argument.

The encoder of the trained model that has learned the specifics of the language can be saved and later used for other downstream tasks (classification/regression).

**Step 3: Creating the Regressor/Classifier**

In the first step, a TextClasDataBunch is created using the vocabulary of the pre-trained or fine-tuned LM. This is done to make sure that the LM and regressor/classifier have the same vocabulary. The batch size ‘bs’ argument can be set (32, 64, 128, 256, etc) according to the memory available.

In the second step, a learner “text\_classifier\_learner” is created for the regression/classification task. It takes the databunch created in the first step as input. For both regression and classification tasks, text\_classifier\_learner is used with the only difference that in case of regression the labels are FloatList item and for classification it is CategoryList. The encoder of the pre-trained/fine-tuned model saved in step 2 can be loaded. Then the same procedure can be followed for finding the lr and training the model.

**Datasets**

The dataset used in pre-training and target task fine-tuning is provided in the ‘Data’ folder.

* The folder ‘Data/pre-training’ contains the ChEMBL dataset used for pre-training.
* The folder ‘Data/fine\_tuning’ contains the data for all three reactions used in this study.

**Code**

All the codes are provided as notebooks that can be directly run on Google Colab.

1. The notebook for pre-training the language model is given in the ‘Pre-training’ folder.
2. All the notebooks for LM and regressor fine-tuning are present in the ‘Fine-tuning’ folder.
3. The notebooks for the training of three different **TL**-**m** models as described in the paper is provided in the respective folders. For instance, code for the training of **TL**-**m1** model is present in the folder named ‘Fine-tuning/TL-m1’.

**References**

* <https://doi.org/10.3390/info11020108>
* https://github.com/XinhaoLi74/MolPMoFiT