Text Mining (732A92)

Utilizing NLP transfer learning with ULMFiT in Chemoinformatic

Spyridon Dimitriadis (spydi472)

January 6, 2021

Abstract

This project explores the capabilities of NLP transfer learning on a text representation other than a natural language. In chemoinformatics, the molecules can be represented as strings of characters (SMILES) and they can be treated as a 'chemistry language'. The Universal Language Model Fine-tuning for Text (ULMFiT) is used, pre-trained on an unlabeled big dataset of one million molecules to learn the 'chemistry language'. Then this model is fine-tuned to predict the lipophilicity property of a molecule which is a continuous value. The pre-training part of the model takes around 6 hours and predicts correctly 82% of the time the next token given the previous 50 tokens. After the model is fine-tuned to predict the lipophilicity of a molecule and it achieved root mean squared error (RMSE) 0.79. While a base line model like random forest which uses features by feature engineering from the molecules, achieves RMSE around 0.98. The results shown that using NLP transfer learning on chemoinformatics achieves strong performances that can outperform the models on structured data by feature engineering.

Contents

1 Introduction

Within the last years, transfer learning has shown incredible results in Natural Language Processing (NLP). The first steps were using pre-trained embeddings, for example GloVe[1] or Word2Vec[2]. With this approach, only the first layer of the network was transferred, using the prior knowledge of only one layer, the other layers were trained from scratch. In 2018 the first Language Models[3, 4] were introduced by transferring all the layer weights (probably changing the last ones to address a different task). A language model models the probability of the next word given the previous n words (or given the m previous words and the m next words). The big picture of NLP transfer learning is to pre-train the language model on a general domain dataset and then transfer this knowledge on a specific task by fine-tuning the weights.

The above NLP techniques can be applied on every 'language'. In chemoinformatics, the molecules can be represented as strings of characters, this could be considered as the 'language of chemistry'. An efficient way to represent them is with SMILES strings (see Data section). Since there is limited amount of labeled data in chemistry, an NLP model could be pre-trained on a self-supervised way on unlabeled data and then transfer this knowledge to a more specific task. The specific tasks could be the prediction of a property of the molecules. In the drug discovery procedure there are a lot of molecule properties that we want to know in order to understand their effects. For example, we want to know how a drug dissolves in fat (lipophilicity property) or how it dissolves in water (solubility property). Thus, we can predict these properties for the molecules of a potential drug, which will save time and money on the drug discovery procedure.

In this project, we will use the SMILES string notation of molecules and the ULMFiT model to predict the lipophilicity of each molecule. ULMFiT is an LSTM network with some heuristics which will be explained in the theory section. First, we will pre-train the language model ULMFiT on one million SMILES in self-supervised learning. Then the language model will be fine-tuned in the specific domain lipophilicity dataset again in self-supervised way. Finally the language model will be transformed to a regression model, dropping the last layer and adding a linear layer, to predict the lipophilicity of each molecule.

2 Theory

2.1 Long Sort-Term Memory network

Humans normally process written text sequentially, capturing the dependence of a word with the previous ones. Language can be processed as a sequence that unfolds in time. Traditional neural networks cannot capture the past information of a word, thus the Recurrent Neural Networks (RNN) address this issue of sequence processing. RNNs are basically neural networks with loops in them; using multiple copies of the same network. Improvements of RNNs are published, in order to address the problem of exploding/vanishing gradients[5]. We can understand vanishing gradient as if we have a really long sentence; so the error has a long way to back propagate through time and it vanishes before it reaches the beginning of the sequence to modify the earlier computations. An improvement of RNN model is the Long Sort-Term Memory (LSTM) networks[6] which are capable of learning long and sort term dependencies. LSTMs introduced the most successes of RNN architecture in NLP tasks. One LSTM network uses four interacting feed-forward layer, so called gates, to manage the long and sort dependencies of the sequence.

In the equations[1] bellow we use $< t >$ to denote time step t, $\widetilde{c}^{<t>}$ is the candidate value for the memory cell at time step t and $c^{<t>}$ is the memory cell at time step t. The notation $[h^{<t-1>}, x^{<t>}]$ denotes the two vectors $h^{<t-1>}$ (the hidden state at time step t-1) and $x^{<t>}$ (the input vector at time step t) stuck one on top of the other to create a long vector. Using the notation of the two stacked vectors makes easier to see the feed-forward layers. The parameters to learn are the four weight matrices W_c, W_i, W_f and W_o with their corresponding biases b_c, b_i, b_f and b_o. On the Figure 1, we can see the unrolled version of an LSTM. The top horizontal line corresponds to the cell state $c^{<t>}$ which is the key concept of the LSTM. Using the cell state the network can easily pass information from one time step to the next. The yellow rectangles are feed-forward layers.

[1]The explanations and equations for the LSTMs are based on: the course of Stanford University "CS 230 - Deep Learning" on the section of RNNs, `https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks`, and on the blogpost "`https://colah.github.io/posts/2015-08-Understanding-LSTMs/`" by Christopher Olah

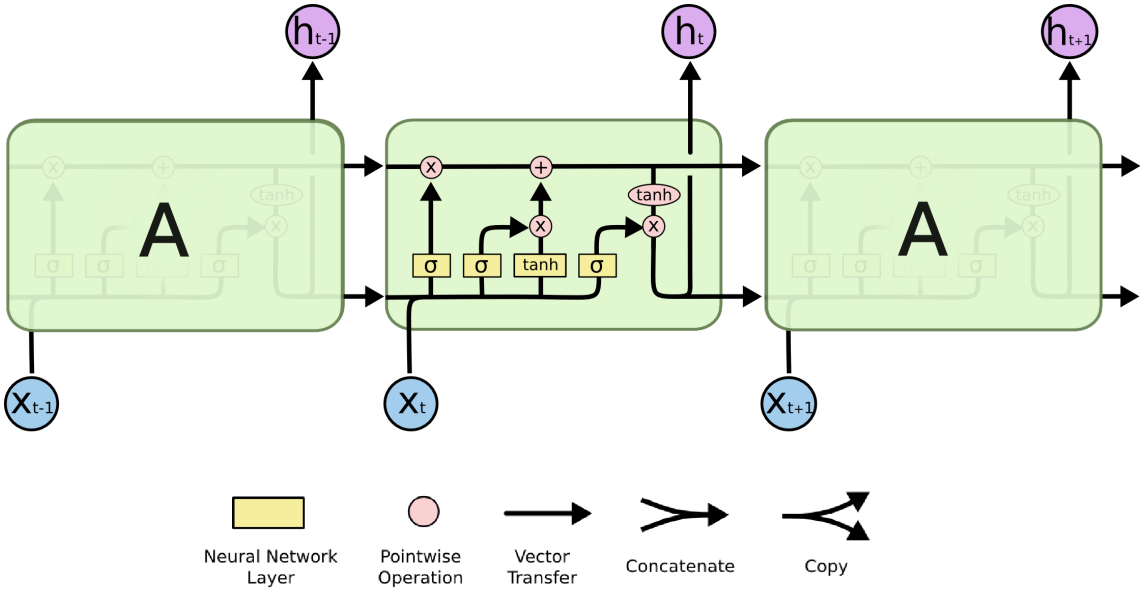


Figure 1: An unrolled LSTM network.
Source: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

- First, the "forget gate layer" $f^{<t>}$ decide the information that will be dropped from the cell state. It takes as input the concatenation of the vectors $h^{<t-1>}$ and $x^{<t>}$ and passes it through the sigmoid function, the output is then between 0 and 1; the closer to 1 is the more information passes through.

$$f^{<t>} = \sigma(W_f[h^{<t-1>}, x^{<t>}] + b_f)$$

- Next the "input gate layer" $i^{<t>}$ define which values will be updated.

$$i^{<t>} = \sigma(W_i[h^{<t-1>}, x^{<t>}] + b_i)$$

- The "update candidate cell layer" creates the candidate cell values $\widetilde{c}^{<t>}$ that could be added to the cell.

$$\widetilde{c}^{<t>} = \tanh(W_c[h^{<t-1>}, x^{<t>}] + b_c)$$

- Now we can update the old memory cell by adding the $f^{<t>} \odot c^{<t-1>}$ with $i^{<t>} \odot \widetilde{c}^{<t>}$, where the symbol $\odot$ denotes the element wise multiplication.

$$c^{<t>} = f^{<t>} \odot c^{<t-1>} + i^{<t>} \odot \widetilde{c}^{<t>}$$

- Finally, the "output gate layer" $o^{<t>}$ will multiplied element wise with the tanh of cell state; in order to filter it to produce the next hidden state.

$$o^{<t>} = \sigma(W_o[h^{<t-1>}, x^{<t>}] + b_o)$$

$$h^{<t>} = o^{<t>} \odot \tanh(c^{<t>})$$

2.2 Averaged SGD Weight-Dropped LSTM

The Averaged SGD Weight-Dropped LSTM (AWD-LSTM)[7] model uses LSTM network and introduce multiple ways to regularize the network. The architecture of the AWD-LSTM consists of an embedding layer of size 400 and three-layer LSTM model with 1152 units in the hidden layer. The embedding weights were initialized uniformly in the interval $[-0.1, 0.1]$ and the LSTM weights were initialized in $[-\frac{1}{\sqrt{H}}, \frac{1}{\sqrt{H}}]$, where H is the hidden size. The regularization techniques that applied to LSTM networks to prevent overfitting and the optimization heuristics from the paper "Regularizing and Optimizing LSTM Language Models"[7] by Stephen Merity, Nitish Shirish Keskar, and Richard Socher are mentioned bellow:

- On the optimization part of the network, the Averaged Stochastic Gradient Descent[8] is used. The average SGD as mantioned by Merity et al.[7] is "similar to SGD, but instead of returning the last iterate as the solution, returns an average of the iterates past a certain, tuned, threshold T".

- The 'weight-dropped' part of the model's name referees to the use of DropConnect[9]. The dropout operation is performed once to the weight matrices before the forward and backward pass, because for multiple time steps the same weights are used then for the whole forward and backward pass the same binary dropout mask is used. The DropConnect is applied on the matrices that correspond to the hidden states ($h^{<t>}$) of the LSTM layers.

- In addition to DropConnect which regularize the hidden to hidden LSTM connections, the variational dropout[10] is used to regularize the input to hidden and hidden to output connections. More specifically, it is used as embedding dropout where the dropout is broadcast on the token vector's embedding and specific tokens will drop for a full forward and backward pass.

- Moreover, 'weight tying' is used, meaning that the weights from input to hidden layer (embedding-hidden) and hidden to output (hidden-softmax layer) are shared. Weight tying reduces the total number of parameters and it is beneficial for the training of the LSTM[11].

- Finally, as means of regularization it is used activation regularization (AR) and temporal activation regularization (TAR)[12]. The idea is that we are not just trying to decrease the loss and the weights but we are also want to decrease the activations. AR penalize the large activations and is defined as $\alpha L_2(m \odot h_t)$, where α is the scaling coefficient to tune, m is the dropout mask and $L_2 = || \cdot ||_2$ is the L_2 regularization. TAR penalizes the large changes in the hidden states and is defined as $\beta L_2(h_t - h_{t+1})$, where β is the scaling coefficient. The AR and TAR loss are applied on the output of the network.

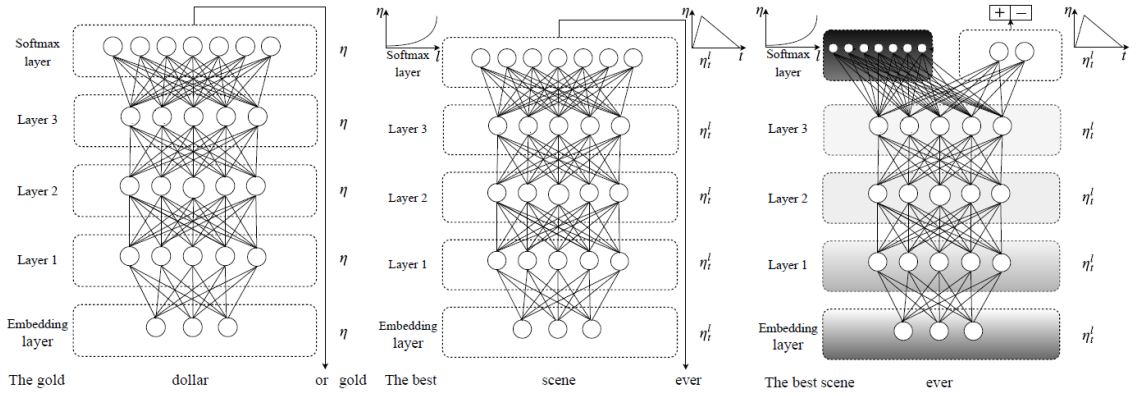
2.3 Universal Language Model Fine-tuning for Text

The paper "Universal Language Model Fine-tuning for Text" (ULMFiT) by Jeremy Howard and Sebastian Ruder[3] was one of the first papers that introduced transfer learning beyond word embeddings in NLP. ULMFiT uses the AWD-LSTM architecture (unidirectional or bidirectional) and adds heuristics on how to train the model. A ULMFiT model consists of three stages:

- First, the language model (LM) pre-traing stage. The LM is trained to predict the next word given the previous m words on a big general-domain corpus of text in a self-supervised way. The loss function of the LM is the cross-entropy loss, since it tries to predict the next word among the words of the vocabulary. In this stage it learns different features of the language on the hidden layers. Usually it is computationally expensive to train but it only needs to be trained once and then be used in different tasks.

- Then, the LM is fine-tuned on a target task to adapt on the task that we want to solve. Probably the target data come from a different distribution than the general-domain data of the previous stage, so fine-tuning the parameters reduces the convergence time of the next stage. As stated in ULMFiT paper, in this stage it is used discriminative fine-tuning and slanted triangular learning rates (STLR)[13]. The discriminative fine-tuning means that each layer of the network is fine-tuned using a different learning rate. The idea behind it is that the early layers capture more general features of the language than the last layers, giving smaller learning rates on the early layers. The STLR suggests that in the early iterations the learning rate should be increased rapidly linearly up to a set maximum value and then it should decrease slowly. It is described in Leslie Smith's paper[13] how STLR helps the model to converge in a local optimum faster. These two ideas are illustrated on the figure 2, in the LM fine-tuning the different layer learning rates are denoted as η_t^l and the learning rate vs iterations plot for STLR is shown above the η_t^ls.

- Finally, the classification model is fine-tuned on target data with their targets. The last layer of LM (the softmax layer) is drooped and two feed-forward layer are added, one layer with relu and the last layer with sofmax for the classification. The last LSTM layer passes to the relu layer the concatenation of the last hidden state, the maxpool of the hidden states and the



(a) LM pre-training (b) LM fine-tuning (c) Classifier fine-tuning

Figure 2: The three stages of ULMFiT as shown in the paper.
Source: `https://arxiv.org/pdf/1801.06146.pdf`

meanpool of the hidden states $h_c = [h_T, maxpool(H), meanpool(H)]$, where $H = h_1, ..., h_T$. The classifier uses discriminative fine-tuning, STLR and gradual unfreezing to maintain the low-level dependencies and adapt the high-level ones. The gradual unfreezing as the word says, gradually unfreeze the layers one by one from the last layer and train for some epochs each time.

2.4 Evaluation metrics

In this project, as evaluation metrics of the language model the accuracy and the perplexity are used. The accuracy simply tells us how many times the model predicted right the next word. This could be problematic in some cases because different words can have the same meaning. The perplexity refers to how complex a sample of text is. A human can make complex sentences (low perplexity score), on contrast to a random generator of words. The perplexity is defined as the exponential average log-likelihood (the cross-entropy loss) of a sequence[2]. Let $X = (x_0, x_1, x_T)$ be the tokenized sequence, the perplexity of X is:

$$PPL(X) = \exp\left\{-\frac{1}{T}\sum_{i}^{T} logp_\theta(x_i|x_{<i})\right\}$$

where $logp_\theta(x_i|x_{<i})$ is the log-likelihood of i-th token conditioned on the previous tokens.

The evaluation metric that has been used on the regression problem is the root mean squared error (RMSE)[3], defined as:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}}$$

where y_i is the target value and $\hat{y}_i$ is the predicted value.

3 Data

3.1 SMILES

The Simplified Molecular Input Line Entry System (SMILES)[14] is the most popular text representation of chemicals. It encodes the molecule structure into a sequence of characters. For example, the oxygen is denoted with O and the symbols $-$, $=$ and $\#$ denote the single, double and

[2]https://huggingface.co/transformers/perplexity.html
[3]http://statweb.stanford.edu/ susan/courses/s60/split/node60.html

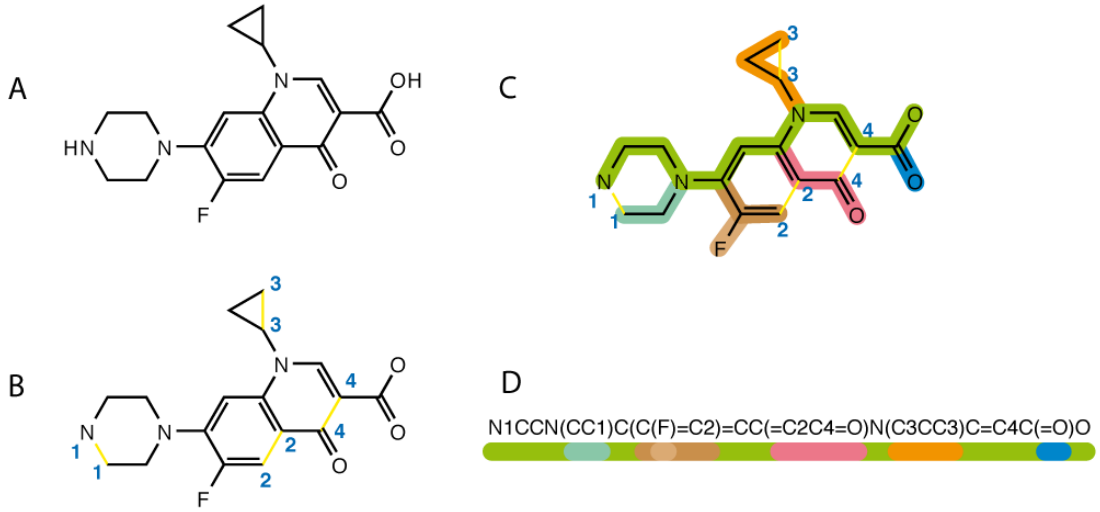


Figure 3: A SMILES string representation of the molecule Ciprofloxacin. In A the molecule is in 2D representation and is converted into a string representation in D.
Source: `https://en.wikipedia.org/wiki/File:SMILES.png`

triple bonds. There are publicly-available datasets that could be used like ZINC[4], ChEMBL[5] and PubChem[6] which contain millions of SMILES.

In this project we will use ChEMBL23[15] dataset to train the Language Model (LM) on a self-supervised way. ChEMBL 23 contains more than 1.5 million SMILES strings on which we will train the LM to predict the next atom. More specifically, we will model the probability of occuring an atom given the previous m atoms.

Afterwards, we will use Fine-Tuning to train the LM on a specific task and convert the LM to a Regressor by keeping the encoder part of the network and adding the last feed forward layers. The idea of Quantitative structure–activity relationship (QSAR) model will be used. The QSAR models are classifiers or regressors that predict different properties of a molecule[16]. In this project we will predict the lipophilicity of a molecule, which refers to the ability of a molecule to dissolve in fat (oils, lipids).

In these last two stages (LM Fine-tuning and Regressor Fine-tuning) we will use the Lipophilicity dataset from deepchem[7].

3.2 Tokenization of SMILES

In the literature exists different approaches to tokenize the SMILES strings. We will use the character-level tokenization with some additional features. Each character will be a token and (1) 'Cl','Br' are two-character tokens, (2) the characters that are encoded between brackets; considered as tokens (e.g. '[nH]', '[C@@H]', etc.). This is considered an atom-wise tokenization of the SMILES. Finally, we included the '[BOS]' begin of smiles token, the '[PAD]' padding token and the '[UNK]' unknown token. For example, the Chalcogran molecule: "CC[C@H](O1)CC[C@@]12CCCO2" is tokenized as: " 'C','C','[C@H]','(','O','1',')','C','C','[C@@]','1','2','C','C','C','O','2' ".

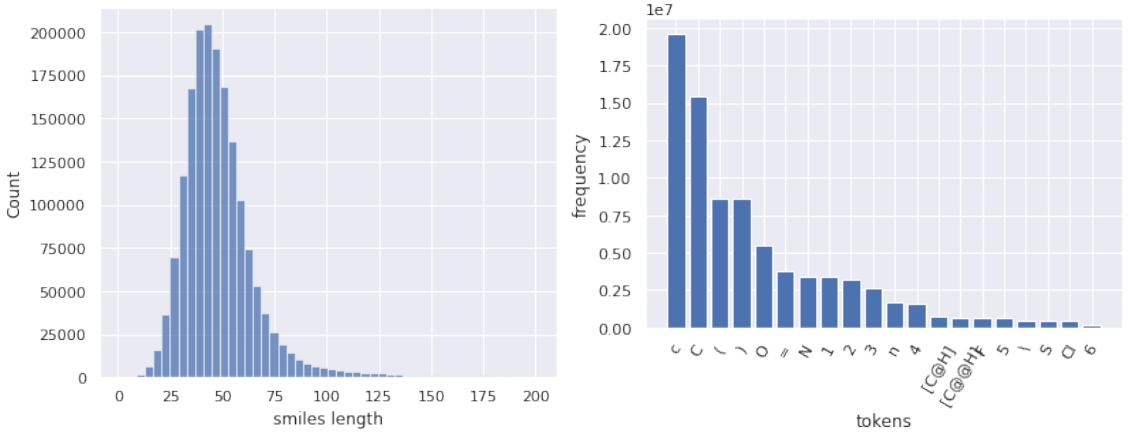
3.3 Data augmentation

SMILES strings augmentation was introduced in 2017 by Esben Bjerrum in "SMILES enumeration as data augmentation for neural network modeling of molecules"[17]. One molecule can have multiple SMILES representation, by simply changing the order of the atoms. In QSAR models the SMILES augmentation has shown that increased the performance[17].

[4]https://pubs.acs.org/doi/abs/10.1021/acs.jcim.5b00559
[5]https://www.ebi.ac.uk/chembl/
[6]https://pubchem.ncbi.nlm.nih.gov/
[7]https://deepchem.io/

3.4 Data statistics

In the ChEMBL 23 dataset the 0.57% of the SMILES have length greater than 200 tokens, so we chose to remove those strings. The average length of a SMILES in ChEMBL23 is 50 tokens, where in an English text it is around 20 words. The vocabulary that is created has 274 tokens. In contrast to an English word vocabulary which could have around 30000 word tokens[18], the SMILES vocabulary size is way smaller. As we see in figure 4b the most frequent token is lowercase 'c' which denotes the aromatic carbon and the capital letter 'C' which denotes the aliphatic carbon[8]. The statistics for the Lipophilicity dataset can be seen in the Appendix section. The length is on average 50 tokens and the most frequent tokens are similar for the two datasets.



(a) SMIELS lengths of ChEMBL23

(b) The 20 most frequent tokens in ChEMBL23.

Figure 4: The left figure shows the SMILES length of the ChEMBL23 dataset, which are less than 200 tokens. On the right plot, the twenty most frequent tokens are shown on the whole ChEMBL23 dataset. Notice that in plot (b) the y-axis is in the power of 10 million.

4 Method

This project is based on the papers "Universal Language Model Fine-tuning for Text Classification" by Jeremy Howard and Sebastian Ruder[3] and the "Inductive transfer learning for molecular activity prediction: Next-Gen QSAR Models with MolPMoFiT" by Xinhao Li and Denis Fourches[19]. The code is based on the work of Xinhao Li[9] and Marcos Santana[10]. The models and training are implemented in the Python packages fast.ai[11] and Pytorch[12]. The data augmentation and SMILES manipulation is performed using the open-source package RDKit[13]. All experiments are run on the Google Colab[14], using a Tesla P100-PCIE-16GB GPU. The source code and details on replicating the results can be found on the corresponding GitHub repository[15].

First, we downloaded the ChEMBL23 dataset as described in Data section. The SMILES strings that have more than 200 tokens are removed. The tokenizer is described in Tokenization of SMILES section. The dataloder of the LM is produced by concatenated all the SMILES into a huge corpus of SMILES one next to the other, separated by the [BOS] token. The independent variable consists of 50 tokens and the dependent variable is the independent shifted by one. In each batch all rows run in parallel, meaning that each row has a SMILES then in the next batch the same rows continues with their SMILES shifted by one. The chosen batch size is 128 and the the 10% of the data is used as validation set. We chose to use 1 million SMILES from 1.7M SMILES in ChEMBL23 to pre-train the LM because of time constrains and limited computational power.

[8] https://www.ics.uci.edu/ dock/manuals/DaylightTheoryManual/theory.smiles.html
[9] https://github.com/XinhaoLi74/MolPMoFiT
[10] https://github.com/Marcosuff/
[11] https://www.fast.ai/
[12] https://pytorch.org/
[13] https://rdkit.org/docs/index.html
[14] https://colab.research.google.com/
[15] https://github.com/spirosdim/Utilizing-NLP-transfer-learning-with-ULMFiT-in-Chemoinformatics

For pre-training the LM on the ChEMBL23, a unidirectional AWD-LSTM is trained ,as described in the theory section, with Adam optimizer and loss function the Cross Entropy. It is trained for 11 epochs, with max learning rate 0.001 using the slanted triangular learning rates for one cycle. Using three LSTM layers with 1152 units it takes more than 10 hours to fully pre-train the LM because of the high number of parameters. So we used 16-bit precision instead of 32 for the parameters to reduce the training time to around 6 hours, this is made easy in the fast.ai library (simply by adding '.to_fp16()' on the training function). The pre-trained LM has 9 million parameters. We saved the pre-trained ML using pickle format, to reuse it.

After we have the pre-trained LM, we fine-tune the LM on the Lipophilicity dataset. This dataset has only 4200 SMILES with their corresponding Lipophilicity measurements, so data augmentation is used to increase the size of the dataset to 92258. The SMILES are augmented as described in data augmentation section and on the dependent variable Gaussian noise is added to the original measurement. More specifically, each SMILES string is augmented 25 times and for each augmented observation in the original Lipophilicity measurement Gaussian noise is added with $\mu = 0$ and $\sigma = 0.3$, as described by Xinhao Li[19]. For the fine-tunig of the LM only the SMILES are used, the dataloader is created as described on the pre-trained LM. The weights are loaded and the last layer is fine tuned for 2 epochs with max learning rate 0.05. Then the whole model is unfreezed and fine-tuned for 8 epochs with max learning rate 0.001. Now the LM is adapted on the Lipophilicity dataset.

Finally, the dataloader for the regression QSAR model is created having the SMILES as independent variable and the Lipophilicity measurements as dependent variable, with batch size 128. The last layer of the LM model is dropped and two feed-forward layers are added to construct the QSAR model for regression, with mean squared error (MSE) as loss function. The fine-tuning of the model is performed with gradually unfreezing the layers and discriminative fine-tuning with learning rates $\eta^{layer-1} = \eta^{layer}/2.6$ [19], as described in the theory section.

The performance of the described ULMFiT model fine-tuned on Lipophilicity data is compared to a random forest (RF)[20] on tabular data produced by feature engineering. Each molecule is described by four features following the work by John S. Delaney[21] on solubility problem. These descriptors are the octanol-water partition coefficient (cLogP), the molecular weight, number of rotatable bonds and the aromatic proportion (number of aromatic atoms over total number of heavy atoms); all calculated using the RDKit package.

5 Results

The ULMFiT on the first stage, pre-training on ChEMBL23 predicts the next token given the previous 50 tokens with 83.9% accuracy and has perplexity 1.6. It approximately takes 7.7 hours (462 minutes) to be trained. On the second stage takes 54 minutes, the fine-tuned ULMFiT on the Lipophilicity data predicts the next token given the previous 40 tokens with 78.8% accuracy and has perplexity 1.88. The third and last stage of the Regressor ULMFiT is gradually fine-tuned for 37.5 minutes and achieves 0.79 RMSE. The baseline model, the random forest on descriptors produced by feature engineering achieves 0.98 RMSE.

Figure 5 shows on the left boxplot the RMSE of the random forest predicting on the test set by training ten times, RMSE 0.983 ± 0.01. The right boxplot shows the RMSE of the ULMFiT Regressor on the test set fine-tuned ten times, RMSE 0.797 ± 0.014. It shows that the lowest RMSE of RF was 0.974 and the lowest of ULMFiT was 0.783 RMSE. Both boxplot results have low variation around the mean, and as observed ULMFiT results have a bit higher variance.

6 Discussion

The results show that NLP transfer learning with ULMFiT on predicting Lipophilicity molecular property can outperform the traditional machine learning algorithms on tabular data. Moreover, now that the weights of the pre-trained language model are calculated and stored, it is easy to use them to predict other properties of the molecules[16].

The paper that followed for this project[19] by Xinhao Li and Denis Fourches describes a number of steps to pre-process the ChEMBL dataset which is used to pre-train the language model. In this project, the luck of these specific chemistry knowledge lead us to skip these pre-processing steps. Moreover the limited time to understand all the details of ULMFiT lead us to limited

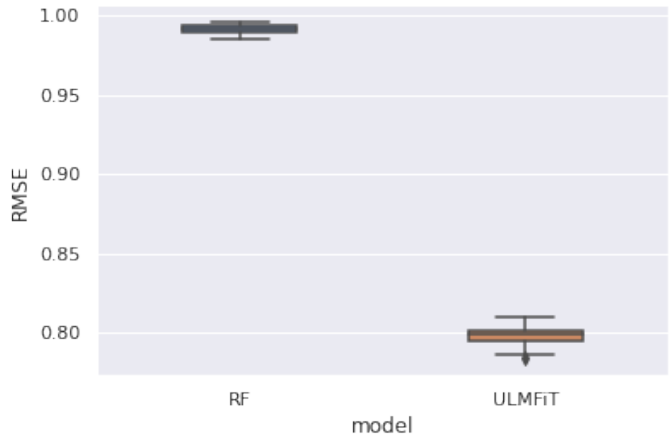


Figure 5: RMSE score from Random Forest on tabular data and ULMFiT on SMILES strings.

hyperparameter search and use of unidirectional LSTM. Concerning the above reasons, we did not reach the performance on the Lipophilicity data shown in the paper, 0.695 ± 0.036.

Additional thoughts that could be investigated in continuation of this project:

- use data augmentation while training. In this project we augmented the SMILES in the beginning before training. This data augmentation could be done in every batch and produce different SMILES, this could work as a regularization as well.
- use bidirectional LSTM or Transformer architectures instead of LSTM. For example, BERT[18] or GTP-2[22] could be used as language model.
- use another way of tokenizing the SMILES strings. A suitable tokenization could be Byte-Pair Encoding (BPE) tokenizer[23], which is used in GTP-2 model.

7 Conclusion

NLP models can be utilized on tasks with text-like data that are not a 'natural language'. In our case, using SMILES a NLP model achieved strong performance on a chemoinformatics task. Moreover, using NLP transfer learning we can see the power of transferring the general knowledge from unlabeled data to solve supervised tasks. Transfer learning can help to achieve high performance having limited amount of data.

References

[1] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[3] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification, 2018.

[4] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.

[5] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks, 2013.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[7] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models, 2017.

[8] Boris Polyak and Anatoli Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30:838–855, 07 1992.

[9] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. volume 28 of *Proceedings of Machine Learning Research*, pages 1058–1066. PMLR, 17–19 Jun 2013.

[10] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. 12 2016.

[11] Hakan Inan, Khashayar Khosravi, and Richard Socher. Tying word vectors and word classifiers: A loss framework for language modeling, 2017.

[12] Stephen Merity, Bryan McCann, and Richard Socher. Revisiting activation regularization for language rnns, 2017.

[13] Leslie N. Smith. Cyclical learning rates for training neural networks, 2017.

[14] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28:31–36, 02 1988.

[15] Anna Gaulton, Louisa J. Bellis, A. Patricia Bento, Jon Chambers, Mark Davies, Anne Hersey, Yvonne Light, Shaun McGlinchey, David Michalovich, Bissan Al-Lazikani, and John P. Overington. ChEMBL: a large-scale bioactivity database for drug discovery. *Nucleic Acids Research*, 40(D1):D1100–D1107, 09 2011.

[16] Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chem. Sci.*, 9:513–530, 2018.

[17] Esben Jannik Bjerrum. Smiles enumeration as data augmentation for neural network modeling of molecules, 2017.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[19] Xinhao Li and Denis Fourches. Inductive transfer learning for molecular activity prediction: Next-gen qsar models with molpmofit. *Journal of Cheminformatics*, 12, 04 2020.

[20] Leo Breiman. Random forests. *Machine Learning*, 10 2001.

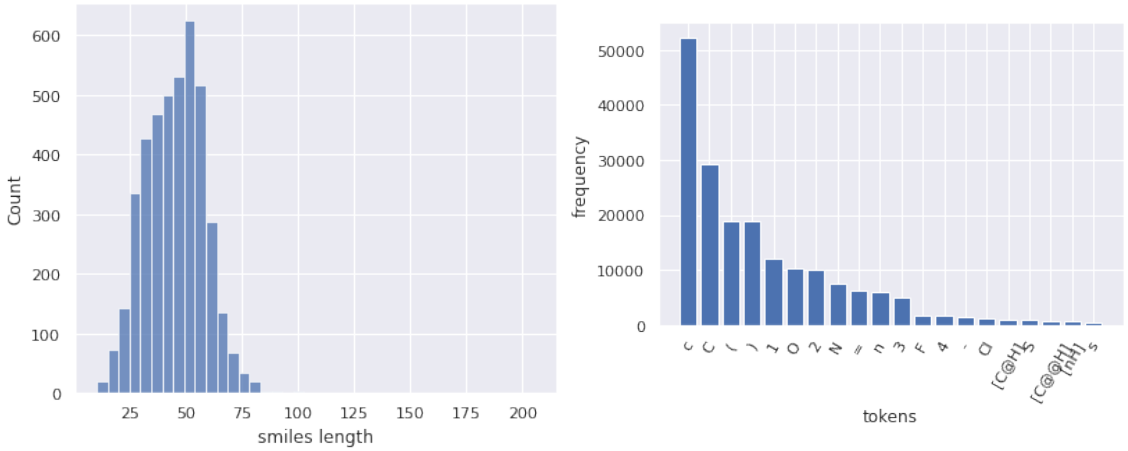
[21] John S. Delaney. Esol: estimating aqueous solubility directly from molecular structure. *Journal of Chemical Information and Computer Sciences*, 44(3):1000–1005, 2004. PMID: 15154768.

[22] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2018.

[23] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units, 2016.

8 Appendix

8.1 Lipophilicity Data



(a) SMIELS lengths of Lipophilicity

(b) The 20 most frequent tokens in Lipophilicity.

Figure 6: The left figure shows the SMILES length of the Lipophilicity dataset. On the right plot, the twenty most frequent tokens are shown on the Lipophilicity dataset.

8.2 Models architecture

```
SequentialRNN (Input shape: ['128 x 40'])
================================================================
Layer (type)         Output Shape         Param #    Trainable
================================================================
LSTM                 ['128 x 40 x 1152',  1,852,416  True
________________________________________________________________
LSTM                 ['128 x 40 x 1152',  5,317,632  True
________________________________________________________________
LSTM                 ['128 x 40 x 400',   1,846,400  True
________________________________________________________________
RNNDropout           128 x 40 x 400       0          False
________________________________________________________________
RNNDropout           128 x 40 x 1152      0          False
________________________________________________________________
RNNDropout           128 x 40 x 1152      0          False
________________________________________________________________
Linear               128 x 40 x 240       96,240     True
________________________________________________________________
RNNDropout           128 x 40 x 400       0          False
________________________________________________________________

Total params: 9,112,688
Total trainable params: 9,112,688
Total non-trainable params: 0

Optimizer used: <function Adam at 0x7f97c225e6a8>
Loss function: FlattenedLoss of CrossEntropyLoss()
```

Figure 7: The language model ULMFit.

SequentialRNN (Input shape: ['128 x 212'])

Layer (type)	Output Shape	Param #	Trainable
LSTM	['128 x 68 x 1152',	1,852,416	True
LSTM	['128 x 68 x 1152',	5,317,632	True
LSTM	['128 x 68 x 400',	1,846,400	True
RNNDropout	128 x 68 x 400	0	False
RNNDropout	128 x 68 x 1152	0	False
RNNDropout	128 x 68 x 1152	0	False
BatchNorm1d	128 x 1200	2,400	True
Dropout	128 x 1200	0	False
Linear	128 x 50	60,000	True
ReLU	128 x 50	0	False
BatchNorm1d	128 x 50	100	True
Dropout	128 x 50	0	False
Linear	128 x 1	50	True

Total params: 9,078,998
Total trainable params: 9,078,998
Total non-trainable params: 0

Figure 8: The QSAR regression model.

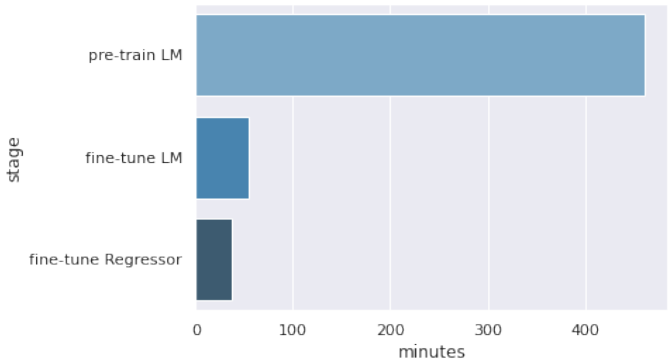


Figure 9: The minutes each stage of ULMFiT did to be trained.