Capstone Project Phase B
Project 22-1-D-16

Department of Software Engineering
Braude College of Engineering, Israel

Universal Language Model Fine-Tuning for Text Classification

In partial fulfillment of the requirements for
Capstone Project in Software Engineering (61998)
Karmiel – June 2022

Supervisors:
Dr. Renata Avros
Prof. Vladimir (Zeev) Volkovich

Group members:
Bradley Feitsvaig Bradley.feitsvaig@e.braude.ac.il
Aviv Okun Aviv.okun@e.braude.ac.il

Table of Contents

1. Introduction **2**
1.1. Terminology 2
1.2. Problem Formulation 2
2. Background and Related Work **3**
2.1 Word Embedding 3
2.2 T-Distributed Stochastic Neighbor Embedding 4
2.3 Language Model Fine-Tuning 4
2.4 LSTM 7
2.5 Related Work 8
3. Achievements **9**
3.1 English Model 9
3.2 Russian Model 10
4. Research Process **12**
4.1 Process 12
4.2 Product 12
4.2.1. Graphical User Interface 14
4.2.2. ULMFiT Algorithm 14
4.2.3. Use Case 16
5. Evaluation/Verification Plan **17**
5.1 Black Box Testing 17
5.2 White Box Testing 18
6. User Documentation **19**
6.1 User Guide 19
6.1.1. General Description 19
6.1.2. System Operation 19
6.2 Maintenance Guide 20
6.2 Environment Specification 22
7. References **22**

Abstract: Training deep neural networks model handling natural language processing (NLP) tasks such as topic classification and sentimental analysis is a tedious process requiring large amounts of time and resources. In our project, an effective transfer learning method is considered to adapt the NLP tools essential for fine-tuning a language model. To this purpose, post-training procedures of several modern methods like transformers are considered with attention to improving the performance of the NLP tasks. The project mainly concentrates on texts composed of Russian and English. The expected outcome is a general outline making it possible to adopt a stylistic model to a current NLP task without training the overall model from scratch.

1. Introduction

1.1 Terminology

Natural Language Processing (NLP) is a subfield of linguistics and artificial intelligence that is involved with the interaction between computers and human language. NLP focuses on computer programs that can process and analyze large amounts of natural language data. Challenges in natural language processing frequently involve speech recognition, natural language understanding, and natural language generation. This project is focusing on the natural language understanding aspect, or more precisely, text classification.

Text classification is a *machine learning* technique that assigns a predefined set of categories to an open-ended text. Perhaps the most common examples of text classificatio*n* are *sentimental analysis*, *topic classification*, *question classification* and *authorship verification*. Sentimental analysis is an automated process that indicates the polarity of a given text (positive, negative, neutral). Companies can use sentimental analysis for a wide range of applications like market research, customer support, workforce analytics, and much more. The topic classification goal is to automatically extract meaning from text by identifying recurrent themes or topics. It is used for structuring and organizing data, such as organizing customer feedback by topic or organizing news articles by subject. Question classification classifies a given question to a set of predefined question categories such as casual, choice confirmation (yes-no questions), hypothetical questions, and factoid (wh-questions). Authorship verification is a research subject that concerned with the question – whether two documents (or books) are written by the same person.

NLP tasks are complex since the text can be an extremely rich source of information and extracting insights from it can be hard and time-consuming. Natural languages, unlike programming languages, evolve as they pass from generation to generation and are hard to pin down with explicit rules. However, thanks to natural language processing and machine learning, which falls under the vast umbrella of artificial intelligence, sorting and analyzing text data automatically became a possibility.

1.2 Problem Formulation

Neural networks models that implement NLP tasks are usually tailored to deal with a specific task. Thus, to build a versatile application, which consequently deals with more than one specific NLP task, there is a need for a variety of deep neural network models. Training *deep neural network* models that can carry out NLP tasks is a tedious process that requires time and resources, thus, there is a necessity for a single model that can switch between tasks with minimal changes to the model itself or a method that can tune the model to deal with different tasks without going through training from scratch or using different models. This is achieved with a state-of-the-art method - Universal Language Model Fine-tuning for Text Classification

(ULMFiT). The project intends to apply the universal language model as described in a paper by Jeremy Howard and Sebastian Ruder to languages like Russian and English and to measure its effectiveness.

2. Background and Related Work

2.1 Word Embedding

To tackle natural language processing (NLP) tasks, a neural network model, must understand the meaning of words, however, words must be represented in such a way that they can be regarded as an input to a model with numerical representation. That is where word embedding comes in hand. *Word embedding* is a term used for the representation of words for text analysis, usually in the form of a vector. On top of representing words as vectors, word embedding must preserve the relation between words so that similar words are represented by vectors with minimal distance. Examples for word embedding algorithms are Word2Vec, Fasttext, or contextually-meaningful embeddings such as ELMo and BERT transformers.
The Advantage of contextually meaningful embeddings over models like Word2Vec is that while each word has a fixed representation under Word2Vec regardless of the context of the word within the sentence, contextually meaningful embeddings produce word representations that are dynamically informed by the context within which the word appears. For example, given two sentences:

"We went to a Shakespeare play yesterday."
"Go play with the dog!"

While Word2Vec would address the word "play" as the same vector, contextually meaningful embeddings such as BERT are distinguishing between the context of the word within the given sentence. Therefore, BERT is embedding the word "play" with different vectors.

2.1.1 Language-Agnostic BERT Sentence Embedding (LaBSE)

A multilingual embedding model is a powerful tool that encodes words and sentences from different languages to a shared embedding space [see, Fig 1], what makes it an ideal tool for NLP tasks like text classification. Previous attempts to build a multilingual model resulted in low accuracy due to the limited model capacity and a poor quality of training data.

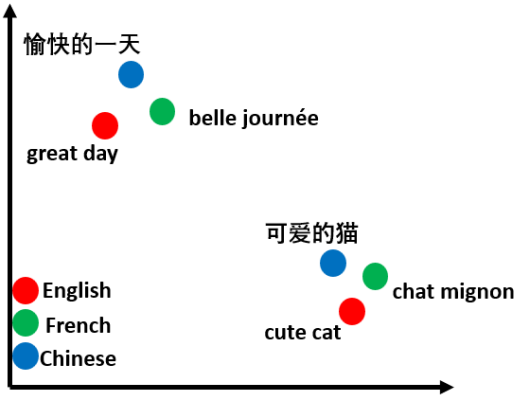

Figure 1: Illustration of multilingual embedding space

Language-agnostic BERT sentence embedding is a BERT embedding model that produces language-agnostic cross-lingual sentence embedding for 109 different languages. The model is trained on 17 billion monolingual sentences and 6 billion bilingual sentence pairs, resulting in a model that is effective even on low-resource languages for which there is no data available during training.

2.2 T-Distributed Stochastic Neighbor Embedding (t-SNE)

T-distributed stochastic neighbor embedding is a statistical method for visualizing high dimensional data by assigning each data point a location in a two- or three-dimensional space.

Given a collection of N high-dimensional objects $(x_1, x_2, \ldots, x_n)$, the goal is to build a low-dimensional map in which the distance between points reflects the similarities in the given high-dimensional data. T-SNE tries to minimize the distance between two probabilities distributions calculated from the neighboring nodes from both the high-dimensional and low-dimensional maps. Kullback-Leibler divergence is used as the cost function for the process.

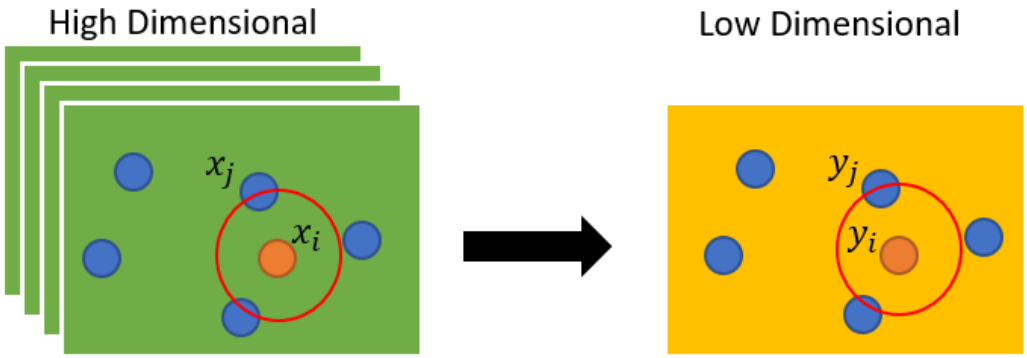

Figure 2: Reducing the dimension of data points while keeping the

The distance between data points is converted into conditional probabilities calculated using the neighbors of each node. The conditional probabilities represent similarities. x_i, x_j, y_i, y_j are considered as data points in the high-dimensional and low-dimensional spaces. The conditional probabilities are calculated:

For the high-dimensional space - $p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)}$

For the low-dimensional space - $q_{j|i} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k \neq i} \exp\left(-\|y_i - y_k\|^2\right)}$

A cost function C is used to minimize the distance between the calculated probabilities.

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

2.3 Language Model Fine-Tuning

Inductive transfer learning refers to the ability of a learning mechanism to improve performance on the current task after learning related concepts from a previous task. This ability has greatly impacted computer vision, but existing approaches in NLP still require specific adjustments for specific tasks and training from scratch. In This project, the language model is initially learned on a general corpus of data in a specific language, this corpus should be large and capture the general properties of the desired language. Afterward, the model is fine-tuned to a specific task. For this specific task, the data corpus should be more specific, such as

Shakespeare's plays or medical articles. The fine-tuning strategy on this model is Discriminative fine-tuning.

Discriminative fine-tuning allows tuning each layer with a different learning rate. In a universal language model, different layers capture different types of information, so it is only natural that different learning rates are applied to each layer separately. Instead of using the same learning rate for all the layers, the language model uses different learning rates for each layer. ULMFiT uses stochastic gradient descent (SGD) to update the model parameters, but in a way that allows every layer to be fine-tuned with different learning rates:

$$\theta_t^l = \theta_{t-1}^l - \eta^l \cdot \nabla_{\theta^l} J(\theta).$$

Where:
η^l – is the learning rate for the l-th layer.
θ_t^l – contains the weights of the l-th layer for iteration t.
$\nabla_{\theta^l} J(\theta)$ – is the gradient regarding the model's cost function for the l-th layer.
$1 \leq t \leq T$ is the iteration number in the model's learning process.
$1 \leq l \leq L$ is a number of layer in the model where L is the total amount of layers.

Stochastic gradient descent is a variant of gradient descent algorithm which is an optimization algorithm that is used while backpropagating through a model and updating its weights. The gradient is calculated with the partial derivatives of the model's cost function to reach the minimum cost function value.
The difference between gradient descent and SGD is that SGD uses a random batch of data points to compute a noisy gradient approximation and uses it to descent step. In contrast to the SGD, the standard gradient descent is computing its vector values with the entire training set, and it can be very slow for a big dataset.
SGD calculation:

$$\theta_t = \theta_{t-1} - \eta \cdot \nabla_\theta J(\theta)$$

Where:
η – is the learning rate.
θ_t– contains the weights for iteration t.
$\nabla_\theta J(\theta)$– is the gradient regarding the model's cost function.
$1 \leq t \leq T$ is the iteration number in the model's learning process.

However, as mentioned before, ULMFiT uses SGD in a way that allows every layer to be fine-tuned with different learning rates, the different learning rates are calculated with Slanted Triangular Learning Rate (STLR).

Slanted triangular learning rate is a learning rate schedule that can speed up the learning algorithm. The schedule is linearly increasing and then reducing the *learning rate* over time [see, Fig. 1]. Instead of using the same learning rate for each layer, slanted triangular learning rate (STLR) is used. This way, the model's parameters can converge quickly to a suitable region, and then, can be refined to their optimal value according to the task-specific features. With this method, the learning rates are linearly increased and decreased according to the following update schedule:

$$cut = \lfloor T \cdot cut\_frac \rfloor$$

$$p = \begin{cases} t/cut, & if\ t < cut \\ 1 - \dfrac{t - cut}{cut \cdot (\dfrac{1}{cut\_frac} - 1)}, & otherwise \end{cases}$$

$$\eta_t = \eta_{max} \cdot \frac{1 + p \cdot (ratio - 1)}{ratio}$$

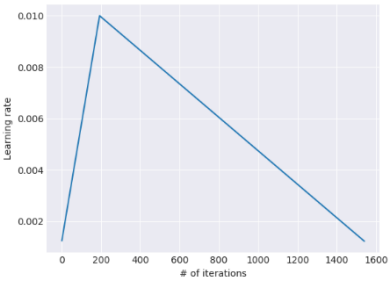

Figure 3: Slanted Triangular Learning Rate Function

T – Is the total number of updates to the learning rate (number of epochs*number of updates per epoch)
cut_frac - Is the ratio between the increasing and decreasing sections of the learning rate graph.
cut - Is the iteration where the learning rate is switching from increasing to decreasing value.
p – Supposed the graph is divided into two sections: t<cut and t≥cut.
For t<cut: p is the fraction of the number of iterations, the learning rate is increasing.
For t≥cut: p is the fraction of the number of iterations, the learning rate is increasing.
ratio – specifies how much bigger the maximum learning rate from the lowest learning rate.
η_t– the learning rate at iteration t.
η_{max}– the maximum learning rate value.

Batch normalization is applied to different chosen layers within the model. When applying batch normalization to a layer, it normalizes the output from the activation function. The batch normalization process is done for batches of data. After normalizing the output from the activation function, batch normalization then multiplies the normalized output by some arbitrary parameter and then adds another arbitrary parameter to this resulting product. This calculation with the two arbitrary parameters sets a new standard deviation and average for the data. These two arbitrary parameters are trained also as part of the model training.

Activation function of a node defines the output of that node given an input.
Rectified linear units (ReLU) function is a linear activation function that outputs the input directly if positive, or, 0 if not.
The function definition:

$$f(x) = \max(x, 0)$$

Sigmoid function is used for mapping a number into small range such as between 0 and 1. Sigmoid function is converting a real value into a number that can be interpreted as a probability.

$$S(x) = \frac{1}{1 + e^{-x}}$$

Tanh function takes any real value as an input and outputs a value between -1 and 1.

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

SoftMax is a function that turns a vector of K values into a vector of K probabilities which sums up to 1. The function works so that negative or small inputs are interpreted into small probabilities values and large inputs, interpreted to large probabilities. It can be used for multiclass classification.
The SoftMax formula:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$\vec{z}$ - is the input vector to the softmax function.
z_i – element of the input vector to the softmax function.
K – number of classes in the multi-class classifier.
Backpropagation through time for text classification (BPTT) is used mainly for large sequences of data. The text document is divided into fixed-length batches. For each batch, the initial state of the model is the final state of the model for the previous batch.

Bidirectional language model is used to predict every word in the sentence given the rest of the words in a way that incorporates both the left and the right context of the sentence. Both forward and backward language models are pretrained, then the classifier for each

language model is fine-tuned independently by using BPTT for text classification, and the average of classifiers predictions.

2.4 LSTM - Long Short-Term Memory

Recurrent neural networks (RNN) are a class of neural networks that allow previous outputs to be used as an input through an internal memory which makes it perfectly suited for machine learning problems that involve sequential data. RNNs are used extensively in natural language processing. The major problem with vanilla RNN is *vanishing gradients*. The gradient carry information used to update the RNN's parameters and when the gradient becomes infinitely smaller, the parameters update become insignificant which makes the task of learning long data sequence nearly impossible. The solution is an upgraded RNN that is called *LSTM*. LSTMs are explicitly designed to avoid the long-term dependency problem. While the repeating module in a standard RNN contains a single layer, the repeating module in an LSTM contains 4 interacting layers [see, Fig. 2].

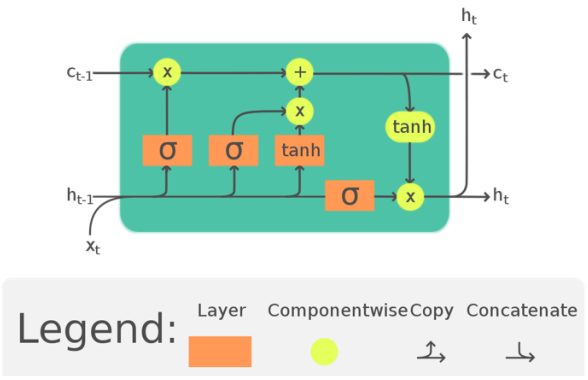

Figure 4: LSTM with 4 interacting layers

The key to LSTM is the cell state C_t. LSTM, can remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information though. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.
The first sigmoid layer called “forget gate layer” which is responsible for removing unwanted information from the previous cell state C_{t-1}.
The second sigmoid layer called “input gate layer” that decides which values are updated in the cell state and the tanh layer creates a vector of new candidate values, $\widetilde{C_t}$, that could be added to the state. These two layers combined, create an update to the cell state.
The final sigmoid layer decides what parts of the cell states are considered as an output for the next iteration. Finally, the cell state is going through a tanh function and is multiplied with the output of the third sigmoid layer. In this project, the type of LSTM used is AWD-LSTM which is a regular LSTM with different tuned dropout hyperparameters.

2.5 GRU – Gated Recurrent Unit

Gated recurrent unit (GRU) is a modified version of its counterpart, the LSTM. Unlike LSTM, GRU combines long and short-term memory into its hidden state and has two gates incorporated in the cell block – update gate and reset gate.

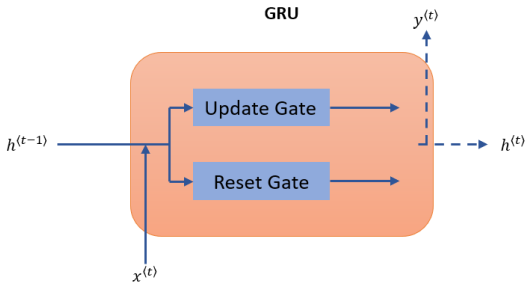

Figure 5: Update gate and reset gate in GRU cell block

The function of the update gate is to know how much of passed memory to retain while the function of the reset gate is to know how much of passed memory to forget.

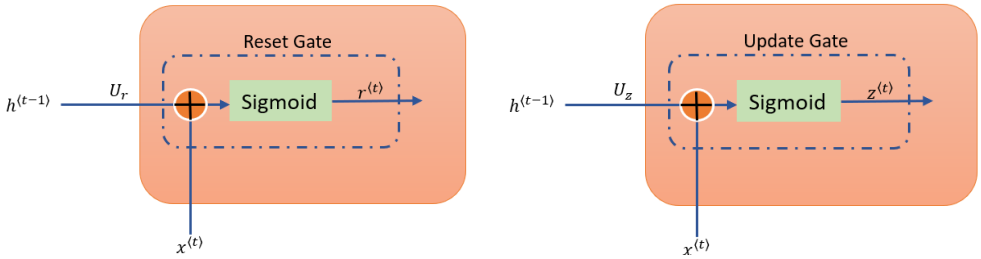

Figure 6: Illustration of the reset and update gates input and output

The reset gate takes hidden state $h^{\langle t-1 \rangle}$ and the current word $x^{\langle t \rangle}$ as an input and performs the following mathematical operation: $w_r x^{\langle t \rangle} + U_r h^{\langle t-1 \rangle}$. Then, a sigmoid activation function is applied on top of it and the final output is: $r^{\langle t \rangle} = \sigma\left(w_r x^{\langle t \rangle} + U_r h^{\langle t-1 \rangle}\right)$.

The update gate takes the same input as the reset gate which is $h^{\langle t-1 \rangle}$ and $x^{\langle t \rangle}$ and performs the same calculation process with one exception – different weights are applied during calculation: $z^{\langle t \rangle} = \sigma\left(w_z x^{\langle t \rangle} + U_z h^{\langle t-1 \rangle}\right)$.

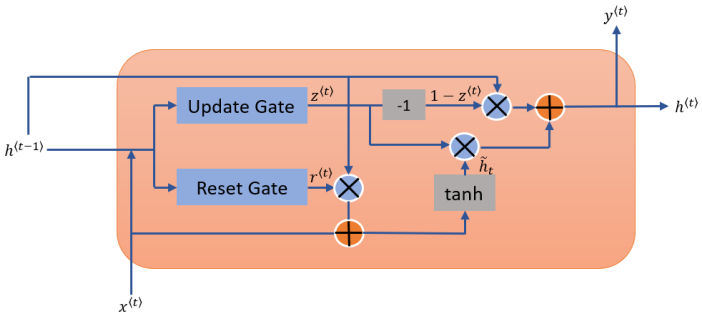

Figure 7: Illustration of a complete GRU cell block

Using both values $z^{\langle t \rangle}$ and $r^{\langle t \rangle}$ values from the gates, a new hidden state $h^{\langle t \rangle}$ is finally produced, as illustrated above.

2.5 Related Work

Transfer learning is an approach in machine learning that focuses on storing a knowledge that is gained from solving one problem and applying it for solving a different but related problem. For example, parameters from a neural network model that can recognize cars can be transferred to a different model that should recognize trucks. This approach is very popular in computer vision because it saves time – the training of a neural network model does not start from scratch.

Multi-task learning is a machine learning approach that tries to learn more than one task simultaneously on the same neural network model. This is the approach taken by Marek Rei (2017) in semi-supervised multitask learning for sequence labeling who add a language modeling task to the main task model. Multi-Task learning is effective when the tasks have shared lower-level features and the amount of data for each task is similar.

Fine-tuning regarding neural networks means taking the weights of a trained neural network model and use them as initialization for a new model that is being trained on data from the same domain. It is used to speed up the training process of a model. Fine tuning has been used successfully to switch between similar tasks like in Sewon Min, Minjoon Seo and Hannaneh Hajishirzi paper – Question Answering through Transfer Learning from Large Fine-grained Supervised Data. Nevertheless, fine tuning has been shown to fail when unrelated tasks are involved.

3. Achievements

A system, alongside an operational GUI, which can perform NLP tasks with two distinct models: an English model and a Russian model. Both models are using the same LaBSE word embedding model that can be applied to different languages and thus, makes it a lot easier in terms of model assembly.

3.1 English Model.

The English model task is a sentimental analysis for Twitter US Airlines. Using the ULMFiT as a general guideline, a smaller LaBSE model fit for 15 languages is used as the embedding layer, thus, producing a general domain language model. The model is then fine-tuned which results in a model that can predict the next word in a sequence of words. At this stage, the model is trained to "understand" the general features of the English language. The dataset used for fine tuning the model is wikitext-2 which is a modeling dataset with over 100 million tokens extracted from the set of verified Good and Featured articles on Wikipedia. Finally, the model is fine – tuned according to the specific task which is sentimental analysis. The data used for the final step is Twitter US airlines sentiment dataset. Both LSTM and GRU are used separately as hidden layers while testing the model performance.

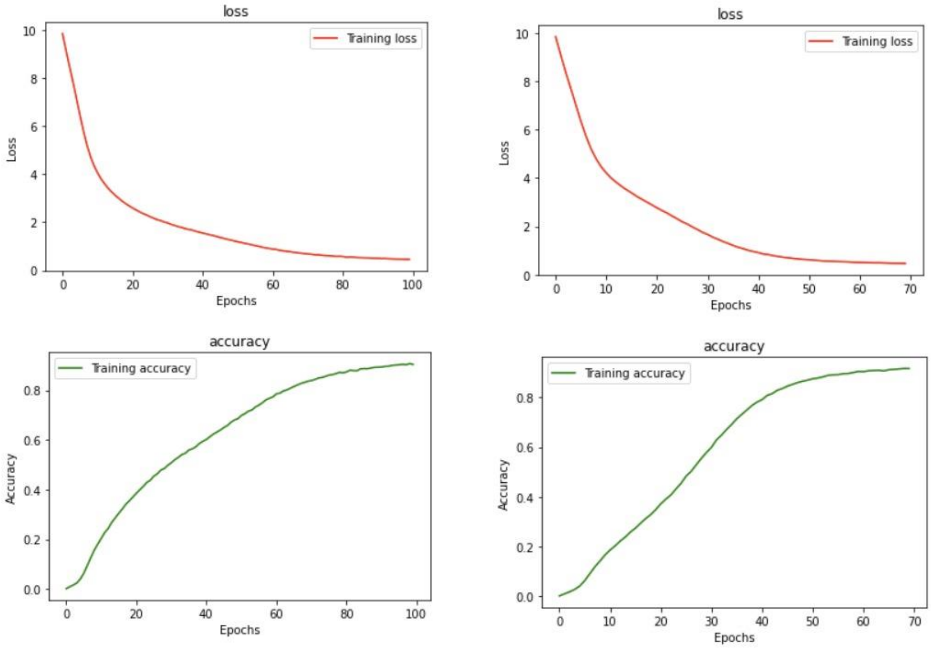

Figure 8: English model - accuracy and loss for the general model fine tuning with LSTM (left) and GRU (right)

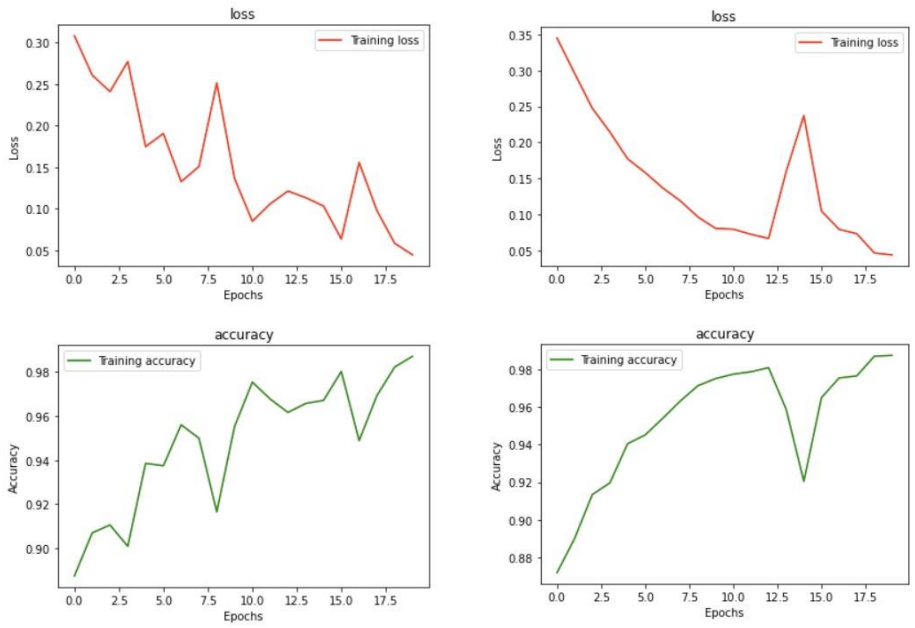

Figure 9: English model - accuracy and loss for the target task classifier with LSTM (left) and GRU (right)

Sentimental analysis for Twitter US Airlines		
Step	Hidden Layer	Train Accuracy
Language model fine tuning	LSTM	90.4%
Target task classifier	LSTM	98.7%
Language model fine tuning	GRU	91.6%
Target task classifier	GRU	98.7%

Figure 10: Sentimental analysis for Twitter Airlines train accuracy

3.2 Russian Model.

The Russian model task is to perform authorship detection on books by the renowned author Mikhail Sholohov whose been rumored to write some of his books with co-authors. The book in question is Тихий Дон (And Quiet Flows the Don) part 1 and 2. Following the ULMFiT principles, LaBSE is used for the word embedding layer to create a general domain model. The model is then trained using Тихий Дон book 1 as a dataset and tested using Тихий Дон book 2. The output data from the model is converted into low dimensional space using t-SNE method and the scatter plot are examined. Both GRU and LSTM are used separately as the intermediate layers to obtain best results possible.

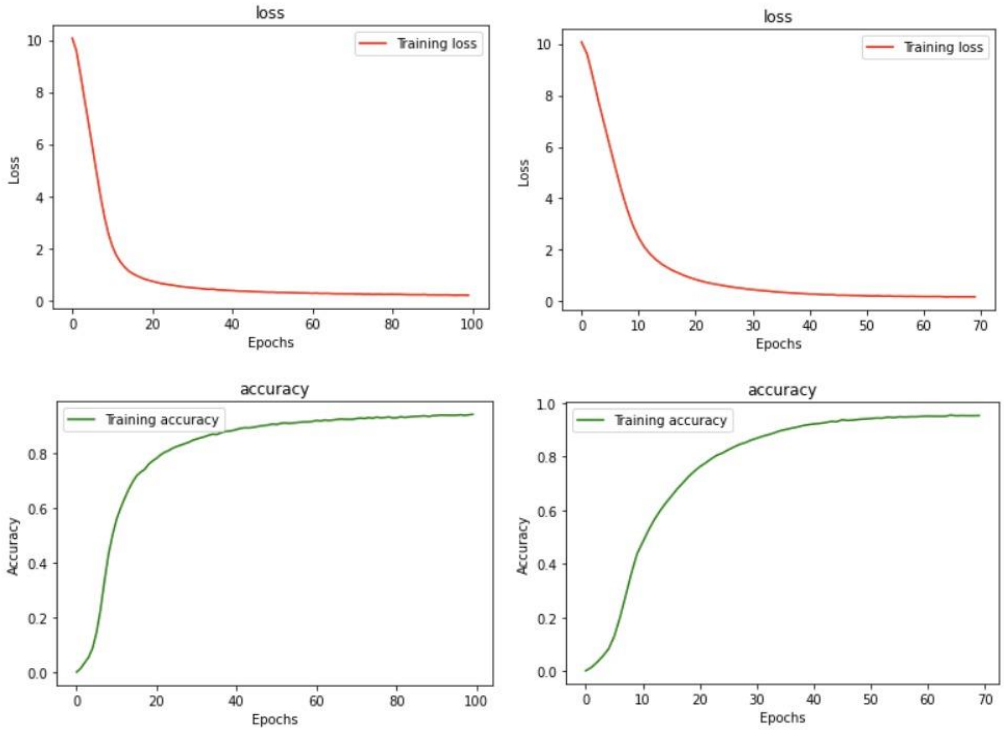

Figure 11: Russian model – accuracy and loss for general model fine tuning with LSTM (left) and GRU (right)

Authorship Detection - Тихий Дон By Mikhail Sholohov		
Step	Hidden Layer	Train Accuracy
Language model fine tuning	LSTM	94.1%
Language model fine tuning	GRU	95.2%

*Figure 12: Authorship detection for **Тихий Дон By Mikhail Sholohov** accuracy results*

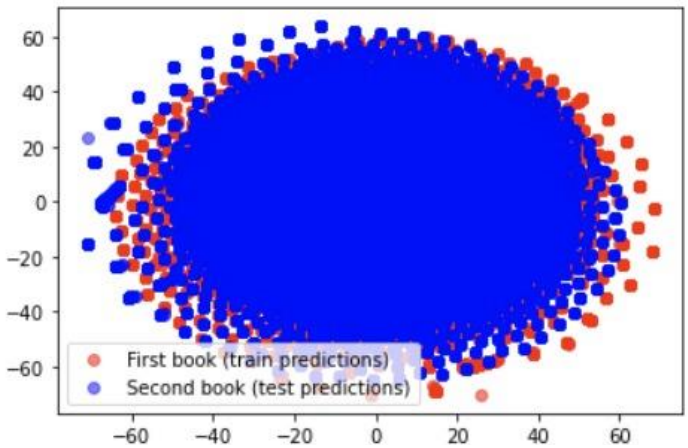

*Figure 13: Scatter plot for **Тихий Дон part 1 and part 2 By Mikhail Sholohov***

4. Research Process

4.1 Process

The research process started by learning more about NLP and the different tasks that are involved. The conclusion was that for each NLP task, the language model is usually trained from scratch on the specific task and the process is time-consuming. Thus, in this approach, developing a multi-task NLP application requires an extensive number of computational resources. To avoid the use of such computational resources, the ULMFiT is suggested.

During the research process, the ULMFiT algorithm was examined with all the components involved. Extensive research was conducted about RNN and more precisely, LSTM and GRU. The project is considering both LSTM and GRU models as part of the overall language model intermediate layers as a mean to achieve better results. The model is using word embedding layers as part of its architecture, and with a recommendation from the supervisors, a transformer is used for it. Research was conducted on BERT and ELMo transformers. The benefits of them are tested during the second phase of the project for the purpose of choosing the most beneficial transformer for the embedding layer. As the project is dealing with two different languages – English and Russian, using two different embedding models (e.g BERT and RuBERT) would result in unnecessary complex code. The solution is a BERT based model that can deal with different languages which is why LaBSE is chosen as the embedding layer for the model. LaBSE can be used for both English and Russian without changing anything in the model itself.

One challenge was coping with the lack of available resources (memory and computational power) for text preprocessing and model training. Instead of using the full LaBSE version which consists of 109 different languages, a smaller LaBSE model with only 15 languages was chosen. To downsize the model memory consumption, instead of using the entire dataset for training, a default of 8000 samples from the dataset was defined. For better hardware this number can be increased easily and the full version of LaBSE can be chosen as embedding layer inside the model building window.

The hardware specification that the model was trained on is a Nvidia Geforce RTX 3070 graphics card with 8 GB available memory and 5888 cuda cores.

For a better understanding of the algorithm components, the mathematical background behind it was studied. A tuning method called discriminative fine-tuning was examined which uses different learning rates for each of the model layers.

The heart of the system is its algorithm that is based on the principles of the ULMFiT and consists of 3 main stages. TensorFlow library is found to be the most effective and easy to use in terms of model assembly and architecture. Regarding the code itself, it was written in PyCharm with the help of pyqt5 for GUI implementation.

4.2 Product

This part describes the main configuration of the system and its uses. The system can be divided into three consecutive parts – the creation and saving of a language model, fine tuning of the language model and specific NLP task prediction.

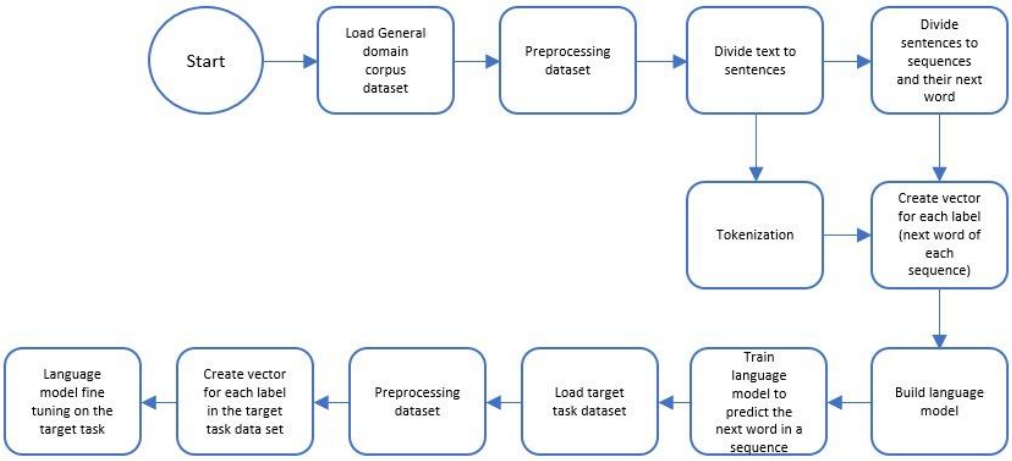

Figure 14: Flow chart of the system

4.2.1 Graphical User Interface

The graphic user interface includes 4 windows. welcome window – with the user help included. Main window – for the execution of fine-tuning on a trained language model and for executing the NLP target task for the fine-tuned model. Build new model window – allows the user to build a custom-made language model and train it on a general corpus dataset. Training analysis window – for examining the model training process.

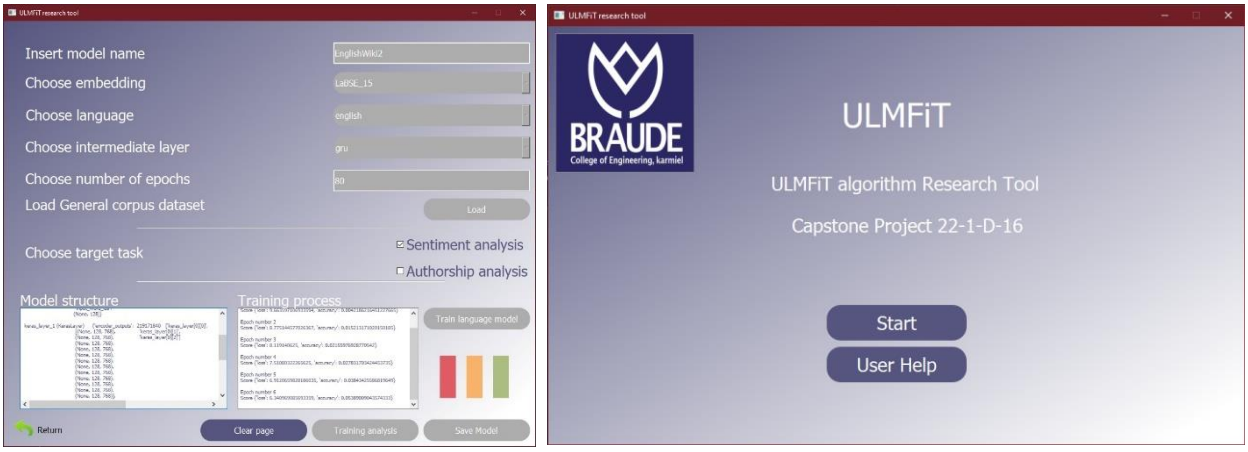

Figure 15: Start window and Language model creation window

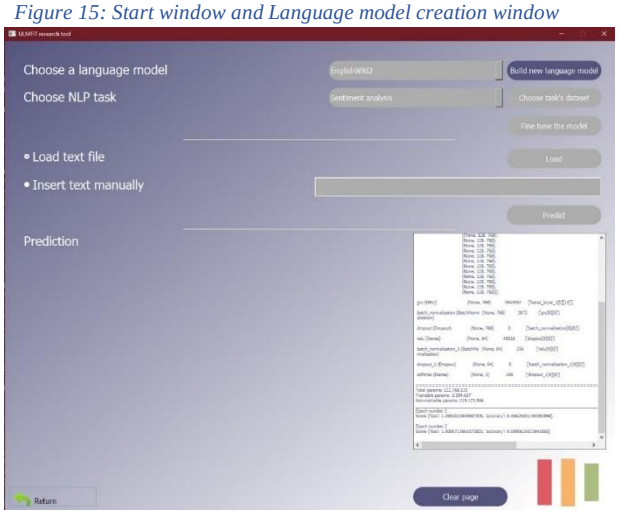

Figure 16: Model fine tuning window

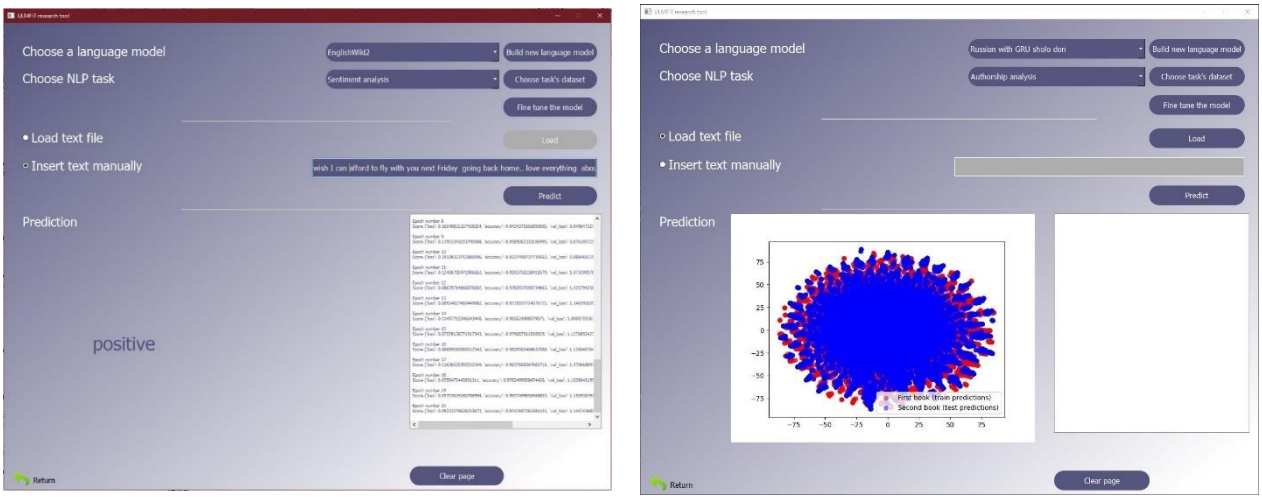

Figure 17: Model predictions

4.2.2 ULMFiT Algorithm

Overview - There are 3 main steps for the ULMFiT algorithm.
The first step is general domain pretraining in which the language model is pre-trained on a large general domain corpus. After the pretraining, the model can predict the next word in a sequence. The second step is target Task Language Model Fine Tuning – the full language model is fine-tuned on a target task data. At the end of this stage, the language model can predict the next word in a sequence of words much like the first step, however it is tuned to match the features of the task specific data. In the third step, the classifier is fine-tuned on the target task.

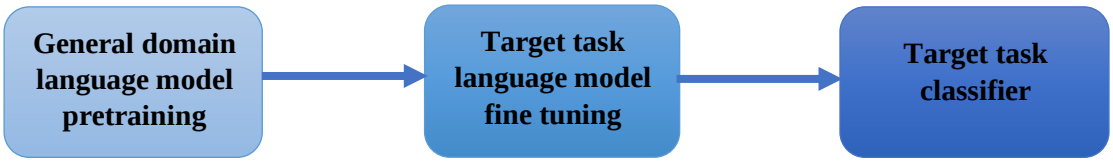

Figure 18: overview of the ULMFiT algorithm

General-Domain Pretraining - In the first step, the language model is pretrained on a large general domain corpus. The general domain corpus needs to be extensive enough so that the model can capture the general features of the language that the model is based on. For instance, for English language, the general dataset could be Wikitext-103 which consists of 28,595 preprocessed Wikipedia articles with approximately 103 million English words.
After the pretraining, the model can predict the next word in a sequence. General domain pretraining is the most expensive part of the learning process but, it is performed only once.

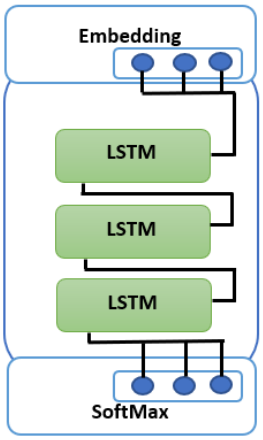

Figure 19: general domain language model architecture

Target Task Language Model Fine Tuning **-** The purpose of the second stage is to fine tune the model on a dataset that is related to the target task. First, the LSTM layers are frozen to focus the fine-tuning process only on the word embedding and the decoder. It is done to ensure that there is no catastrophic forgetting in the hidden layers. Secondly, all the layers are unfrozen, and then, the entire language model is tuned with discriminative fine tuning which means that the learning rate is different for each layer in the model. The learning rate is calculated with slanted triangular learning rate. At this stage, the model is predicting the next word in a sequence much like the first stage, however, it is done in the context of the data of the target task. Finally, the decoder and the SoftMax layers are cut off.

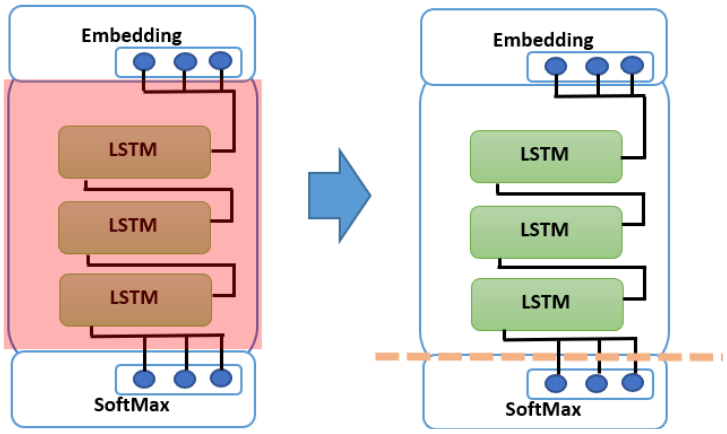

Figure 20: hidden layers of the mode are frozen and then the SoftMax layer is cut off.

Target Task Classification **-** After the second stage, the model must be adjusted according to the targeted task. Two linear blocks are added at the end of the model. Each block uses batch normalization and dropout, with ReLU activation function for the intermediate layer and a SoftMax activation that outputs a probability distribution over target classes at the last layer. The structure of those layers is decided according to the targeted task in hand. The model is frozen and then it is gradually unfrozen to fine tune it while using discriminative fine tuning and slanted triangular learning rates.

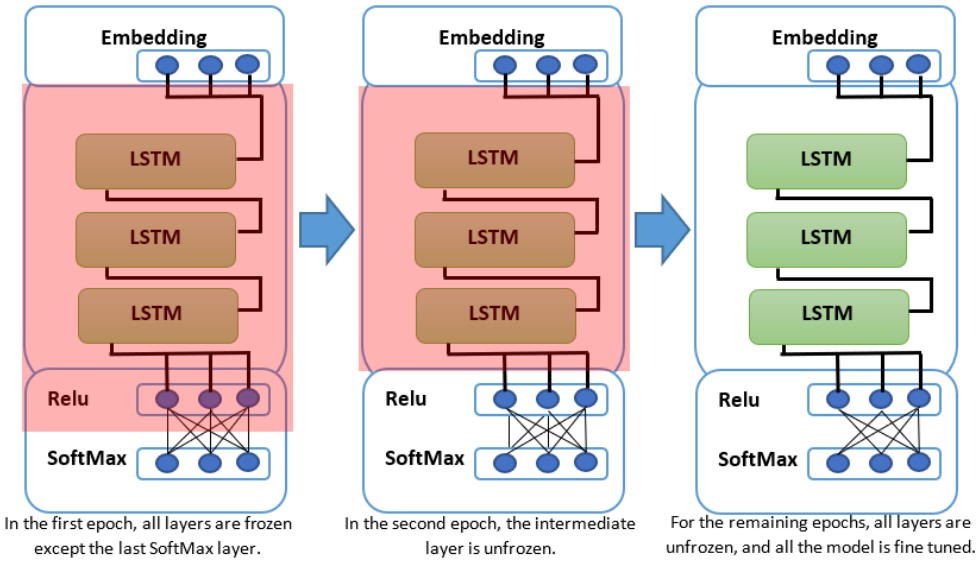

Figure 21: gradual unfreezing the model through epochs.

Final model structure - The finalized model structure that is achieved at the end of the 3 ULMFIT stages consist of 3 main parts: The word embedding, the hidden layers and 2 linear blocks. The word embedding layer, uses word embedding algorithm that converts word to vector. The hidden layers consist of 3 LSTM architectures or more precisely AWD-LSTM. The last part of the model consists of Two linear blocks. Each block uses batch normalization and dropout, with ReLU activation functions for the intermediate layer and a SoftMax activation for the last layer to outputs probability distribution of target classes.

For experimenting purposes, the LSTM layers are replaced with GRU and the possibility that the GRU is preferable for the intermediate layers is considered. GRU was Introduced by Cho, et al. in 2014. Like LSTM [See 2.5.], Its aim is to solve the vanishing gradient problem that comes with a standard recurrent neural network. GRU uses 2 gates, update and reset gate which can decide what information is passed to the output. Moreover, these gates can be trained to choose what data in a sequence should be saved.

4.2.3 Use Case Diagram

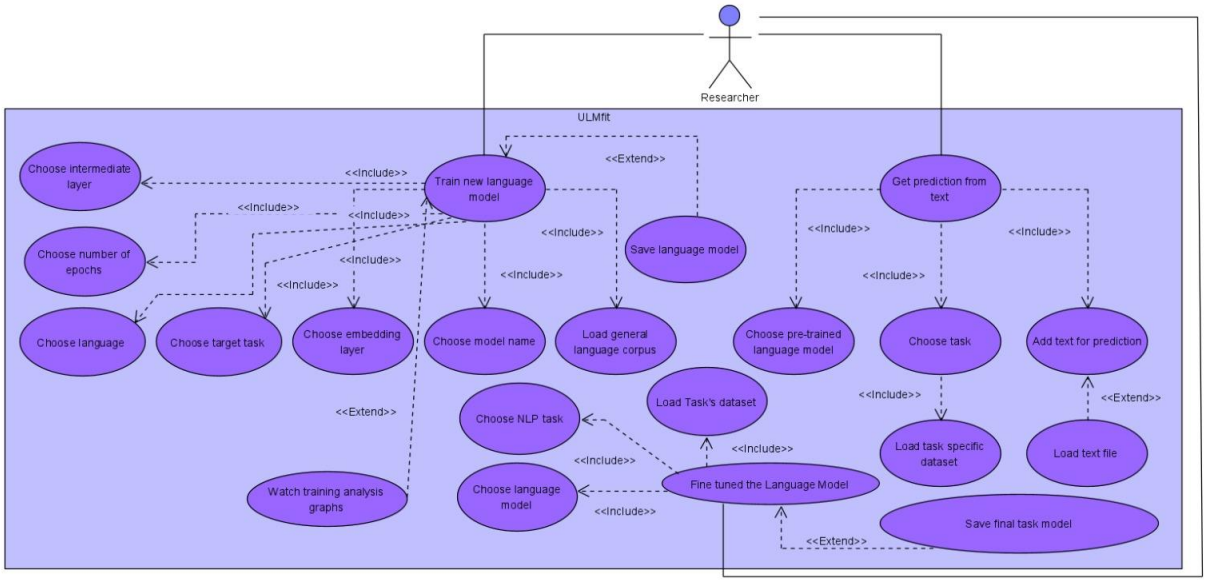

Figure 22: Use case diagram

5. Evaluation/Verification Plan

For the system evaluation and verification, we used two types of testing methods: "black box" and "white box" testing. In black box testing we conduct functional testing. In white box testing we conduct the unit tests with python unittest library.
All testing is done on the already existing 4 model in the project directory:
"English with LSTM wiki2" **,** "English with GRU wiki2" ,"Russian with LSTM sholo don", and "Russian with GRU sholo don".

5.1 Black Box Testing

(1) For getting prediction from text functionality, all sentiment analysis prediction variations were tested by altering the model input for prediction for both "English with LSTM wiki2" and "English with GRU wiki2" models.

Test Case	Input	Results
Case A – positive sentence	"you are one great airline"	POSITIVE
Case B – negative sentence	"how about starting without the robotic response?"	NEGATIVE
Case C – neutral sentence	"I sent in my feedback"	NEUTRAL
Case D – input failure	There is no input	"You have to insert text or load text file to get prediction"

Table 1

(2) For getting prediction from text functionality, the authorship analysis is tested by using different books by Mikhail Sholohov as an input and the cluster formulation is examined.

Test Case	Input	Results
Case A	"And quiet flows the don 2"	2 different clusters, 1 for the training book " And quiet flows the don 1" and the second for the input book.
Case B – input failure	There is no input	"You have to insert text or load text file to get prediction"

Table 2

(3) For training new language model functionality.

Test Case	Input	Results
Case A	All parameters are filled properly.	Training process is shown on the lower half of the window.
Case B	Not all parameters are filled properly.	GUI displays an error message: "You have to fill all parameters for the new language model"

Table 3

(4) For fine tuning a language model functionality.

Test Case	Input	Results
Case A	All parameters are filled properly.	Training process is shown on the lower half of the window.
Case B	Dataset file is not loaded properly.	GUI displays an error message: "You have to load task's dataset file"

Table 4

5.2 White Box Testing

Dataset preprocessing tests functions :

test_divide_text_file_into_sequences_and_next_words – checks if the number of sequences is equal to the number of labels after dividing the raw text.

test_build_data_set_for_sentimental_analysis – checks if the number of sentiment sentences is equal to the number of labels and there are exactly 3 classes of predictions.

test_vectors_for_labels – checks if the returned value is of type "list".

test_build_dataset_for_predicting_the_next_word – checks if the number of sequences is equal to the number of labels after building the dataset for training the model.

Main screen testing (app.py):

test_refresh_page – checks that the window is returned to the default state after pressing the clear button: file_name, task_data_set_file_name and current_model references are None. fine_tune_the_model_button, load_text_file_button, lineEdit, predict_button, insert_text_manually_radioButton and load_text_file_radioButton are disabled. insert_text_manually_radioButton is not checked. lineEdit is clear.

test_refresh_model_task_dict – checks if the LM_comboBox displays all the available language models in the system.

test_start_loading_animation – checks if at the start of a GUI operation, the LM_comboBox,task_comboBox, choose_tasks_dataset_button, fine_tune_the_model_button, predict_button and clear_page_button widgets are disabled.

test_stop_loading_animation - checks if at the end of a GUI operation, the LM_comboBox,task_comboBox, choose_tasks_dataset_button, fine_tune_the_model_button, predict_button and clear_page_button widgets are enabled.

test_task_combobox_activated - checks if the model task combo box is displaying only the relevant tasks according to the selected model.

Build new model screen testing (app.py):

test_clear_page - checks that the window is returned to the default state after pressing the clear button: language_model_name, languag tasks, train_history, accuracy_plot, loss_plot,

model_dir_name_to_save, current_model and general_corpus_dataset widgets are set to "None".
sentiment_checkBox and autorship_checkBox widgets are not checked.
insert_model_name_line_edit and choose_number_of_epochs_line_edit widgets are clear.
save_model_button and training_analysis_button widgets are disabled.

test_choose_embeddings_combo_box_activated – checks if the choose language combo box is displaying only the relevant languages according to the selected embedding.

test_start_loading_animation - checks if at the start of a GUI operation**,** the insert_model_name_line_edit, choose_embeddings_comboBox, choose_language_comboBox ,choose_intermediate_layer_comboBox, choose_number_of_epochs_line_edit ,load_general_corpus_dataset_button, train_language_model_button and clear_page_button widgets are disabled.

test_stop_loading_animation - checks if at the end of a GUI operation, the insert_model_name_line_edit, choose_embeddings_comboBox, choose_language_comboBox ,choose_intermediate_layer_comboBox, choose_number_of_epochs_line_edit ,load_general_corpus_dataset_button, train_language_model_button and clear_page_button widgets are enabled.

6. User Documentation

6.1 User Guide

6.1.1 General Description

The ULMFiT research tool main goal is to examine the operation of the ULMFiT algorithm. With the help of the tool, the algorithm can be applied to different languages and NLP tasks. For the different NLP tasks that are available in the system, the algorithm is executed according to the need of the specific task, thus all the algorithm stages can be examined separately.

There are two main stages in which the algorithm can be executed – the training of a general domain language model and the fine tuning of the language model for a specific NLP task. A different window is associated with each stage of the algorithm.
The main window is for fine tuning of an existing language model. The second window (build new model window) is for the purpose of creating a custom-made language model based on the ULMFiT principles for further research.

The tool is oriented towards researchers who want to examine further the ULMFiT application on different models and languages.

6.1.2 System Operation

Building a new language model window – on the "building new language model" window, a model name must be inserted which will be used when saving the model. A word embedding layer is chosen for the model, together with the model language (English or Russian), the intermediate layers (GRU or LSTM) and the number of epochs for which the model will be trained. A general corpus dataset in the form of a txt file needed to be loaded and the target task is chosen whether its Authorship analysis or Sentiment analysis. In the end of

the model creation process, the "Training analysis" button will be enabled and will lead to a separate window which will show available graphs of the training process.

Training analysis window– In this window, the graphs of the training process will be shown.

Main window – The first section of the main window is model fine tuning. Choose a model from the list of the available trained language models and a desired NLP task from the list of available tasks. In case of sentimental analysis, load a specific target task dataset, **it must be** in the form of a CSV file (an existing CSV file is located inside the datasets directory named "Twitter US Airlines Sentiment dataset.csv") – the header for the tweets should be named "text" and for the label(sentiment) the header should be named "sentiment". The sentiments should be one of the following: "positive", "neutral" or "negative". In the case of authorship analysis, the language model is already satisfying the requirements of the specific task, thus, it is not necessary to proceed with the next stage of the ULMFiT algorithm and the "Choose task's dataset" button will be disabled. In case a model which is already fine-tuned is selected, the "Fine tune model" button will only load an already tuned model and the fine tuning will not execute. To fine tune the model, click on "Fine tune the model" button. In the fine-tuning process, the model architecture and tuning progress is shown on the lower-right section of the window.

The second section of the main window is the execution of the NLP task. After fine-tuning the language model, there are two possibilities: (1) load a text file with the "Load" button in the format of a ".txt". The file should have the content of a book for the authorship analysis task or a sentence for the sentiment analysis task. (2) Insert text manually with the "Insert text manually" bar. After the text is loaded, click on the "Predict" button. The system will show the prediction results on the lower-left side of the window.

****Pay attention that the loaded files should be in the same language all the way through the process according to the language of the model.***

6.2 Maintenance Guide

For updates and further research in terms of new types of NLP tasks and model architecture, the shelve_db.py should be altered. The purpose of the shelve_db.py is to get, set and update shelve files which are used as a local database for the ULMFiT research tool. The shelve module in Python's standard library is a simple yet effective tool for persistent data storage when using a relational database solution is not required. The content of the shelve file is in the form a dictionary.

The update for the shelve_db.py file is set according to the following rules:

(1) Add new word embedding models:

- Class name: "ModelShelveDb"
  - Method name: "set_url_for_embedding"
  - Under sh['url_for_embedding']**,** add a dictionary in the following form "Embedding name": [URL for the text preprocessing, URL for the embedding encoder].
  - Example: "Bert": ["https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3",

"https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/4"],

- Class name: "AppShelveDb"
  - Method name: "set_embedding_options_dict"
  - Under sh['embedding_options_dict']**,** add a key and value in the following form: "Embedding name": [list of available languages in the system].

- Example: "LaBSE_15": ["english", "russian"]

(2) Add new sentiment:

- Class name: "DatasetPreprocessShelveDb"
  - Method name: "set_sentiment_dict"
  - Under sh['sentiment_dict']**,** add a key and a value in the following form "sentiment":number
  - Example: "positive":2
- Class name: "MiscMethodsShelveDb"
  - Method name: "set_sentiment_dict"
  - Under sh['sentiment_dict']**,** add a key and a value in the following form number:"sentiment"
  - Example: 2:"positive"

(3) Add new language:

- Class name: "DatasetPreprocessShelveDb"
  - Method name: "set_regex_for_language"
  - Under sh['regex_for_language']**,** add a key and a value in the following form "language":'[characters that will stay in the text]'
  - Example: "english": '[^a-zA-Z .]'
- Class name: "AppShelveDb"
  - Method name: "set_embedding_options_dict"
  - In the wanted embedding under sh['embedding_options_dict']**,** add a new language in the following form: "embedding": ["english", "russian","new language"]
  - Example: "LaBSE_15": ["english", "russian","arabic"]

***At the end of the desirable changes, run shelve_db.py and the database will be updated.**

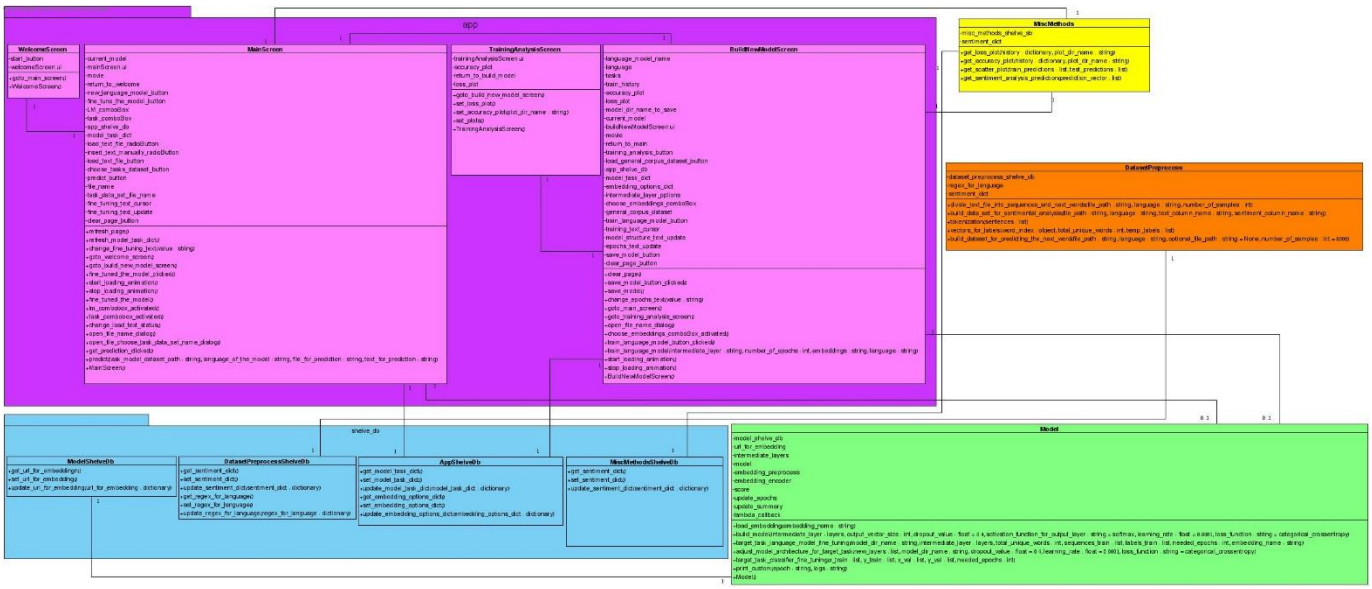

Figure 23: Class diagram

6.3 Environment Specification

Using the tool requires configuration to the environment. To run the application, please go through the following steps:

1) Confirm that the latest Anaconda version is installed, and the latest Python environment (configured with Anaconda) is available.
2) Create a new environment using Anaconda.
3) Run "install.bat" which is attached to the project directory. **In case of a failure during installation, install manually the following libraries:**
    - pyqt5
    - sklearn
    - pandas
    - nltk
    - keras
    - tensorflow
    - matplotlib
    - tensorflow_hub
    - tensorflow_text

    ***For the nltk library, an additional download is required:**
    On command prompt type –
    Python
    >>import nltk
    >>nltk.download('stopwords')
4) Navigate using command prompt to the project directory and run the "app.py" file: **python app.py**

7. References

[1] Jeremy Howard, Sebastian Ruder.2018. Universal Language Model Fine-tuning for Text Classification. arXiv preprint arXiv:1801.06146v5.
[2] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017a. Regularizing and Optimizing LSTM Language Models. arXiv preprint arXiv:1708.02182.
[3] Marek Rei. 2017. Semi-supervised multitask learning for sequence labeling. In Proceedings of ACL 2017.
[4] Sewon Min, Minjoon Seo, and Hannaneh Hajishirzi. 2017. Question Answering through Transfer Learning from Large Fine-grained Supervision Data. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Short Papers).
[5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. arXiv preprint arXiv:1412.3555v1.
[6] Fangxiaoyu Feng , Yinfei Yang , Daniel Cer, Naveen Arivazhagan, Wei Wang. Language-agnostic BERT Sentence Embedding.
[7] Laurens van der Maaten, Geoffrey Hinton. Visualizing Data using t-SNE.