Capstone Project Phase A
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 – January 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

Abstract: Training deep neural networks model handling natural language (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,* and *question classification*. 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).

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.

2.2 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:

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

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

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

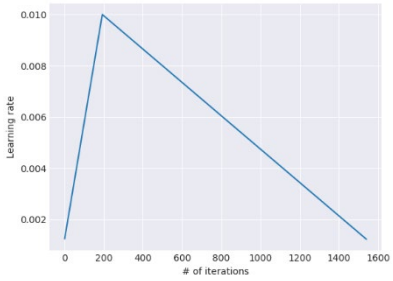

Figure 1: Slanted Triangular Learning Rate Function

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.

2.3 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].

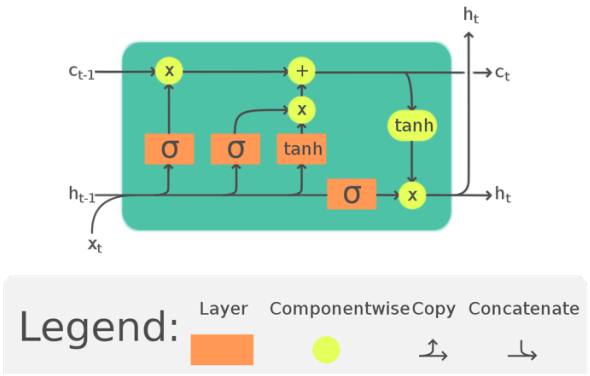

Figure 2: LSTM with 4 interacting layers

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.

2.4 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. Expected Achievements

3.1 Outcomes

3.1.1 Research Tool

A research tool that allows to examine the effectiveness of the ULMFiT as Jeremy Howard and Sebastian Ruder described in their paper. However, the suggested model is implemented on English non-English languages. For non-English language, Russian text is used and for English, texts from Shakespearean plays are used. Two language models are considered in our project. One that is capturing the Russian language features, and a second model for the English language. The way the algorithm works is as follows: The first step is general domain pretraining in which the language model is pre-trained on a large general domain corpus. Afterward, the task-specific dataset is cleaned (e.g links are removed) and is split into train and test sets. Then, the target task language model is fine-tuned on target task data. In the third and final step, the classifier is fine-tuned on the target task [See 4.2.1 for a more detailed algorithm]. In addition, for further research purposes, the tool is allowing to build new language models with modified components without engaging with the written code itself.

3.1.2 Application

The application is implemented in python, and the GUI is be implemented with pyqt5 Python library. The user can choose from more then one NLP tasks (e.g topic classification, sentimental analysis) in both the Russian and the English models and get a prediction for the text which the user has set as an input. Furthermore, the user can create new language models for further research.

3.2 Success Criteria

- ✓ The ULMFiT algorithm is implemented successfully – the model can switch between tasks.
- ✓ Functional and easy to use research tool that allows to perform experiments efficiently.
- ✓ Successfully implementing the ULMFiT algorithm with modifications to the model such as using GRU instead of LSTM and using transformers like BERT and ELMo as word embedding layers. Thus, achieve better results using the ULMFiT.

3.3 Unique Features

- A language model that can switch between NLP tasks without the need to train the model from scratch.
- Using ULMFiT on the Russian language.
- Using ULMFiT on the English language.
- An ability to create new and modified language models without engaging with the code.

3.4 Challenges

- Finding dataset that is used as a large general domain corpus and is extensive enough so that the model can capture the general features of the language both for Russian and English.
- Using different components for the model structure as described in the paper – transformers (e.g BERT, ELMo) for the word embedding layer and GRU instead of LSTM.

4. Research Process

4.1 Process

We started our research process by learning more about NLP and the different tasks that are involved. We learned 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 amount of computational resources. To avoid the use of such computational resources, the ULMFiT is suggested.

During our research, the ULMFiT algorithm was examined with all the components involved. We conducted extensive research 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. Our model is using word embedding layers as part of its architecture, and with a recommendation from our supervisor, a transformer is used for it. We conducted research 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.

For a better understanding of the algorithm components, we have studied the mathematical background behind it. We learned about a tuning method that is called discriminative fine-tuning which uses different learning rates for each of the model layers. The learning rates are calculated by a function called slanted triangular learning rate (STLR). In the next phase of the project, the consequences of using it while modifying its parameters are explored for the purpose of achieving better results.

While searching for related works, we noticed that most of them used a pretrained model as a base to their work. In our project, it is not possible to use pretrained model because we plan on experimenting with different components in our model. Thus, both of our models

are trained from scratch in the first stage of the algorithm, what takes an extensive amount of computational resources to pre-train the general language model.

4.2 Product

The outcome of the research is a software application that can implement NLP tasks while using one deep learning model for Russian and one modern English. The models can switch between tasks by using the ULMFiT algorithm.

4.2.1 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.

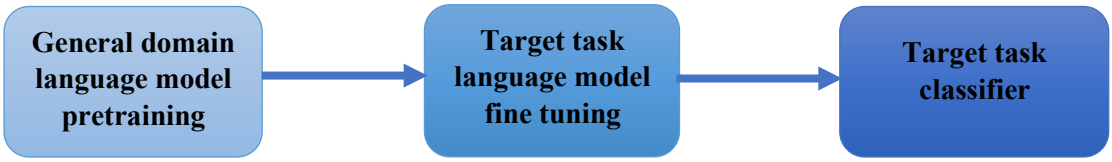

Figure 3: overview of the ULMFiT algorithm

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.

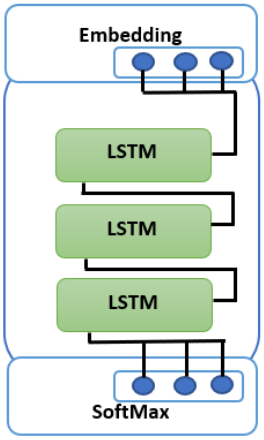

Figure 4: general domain language model architecture

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.

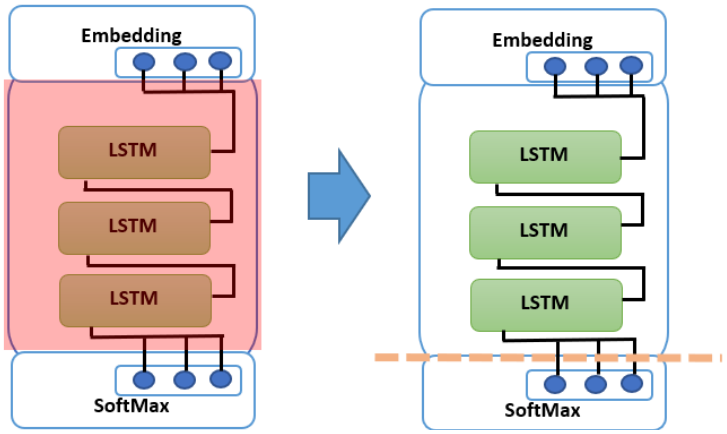

Figure 5: hidden layers of the mode are frozen and then the SoftMax layer is cut off.

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.

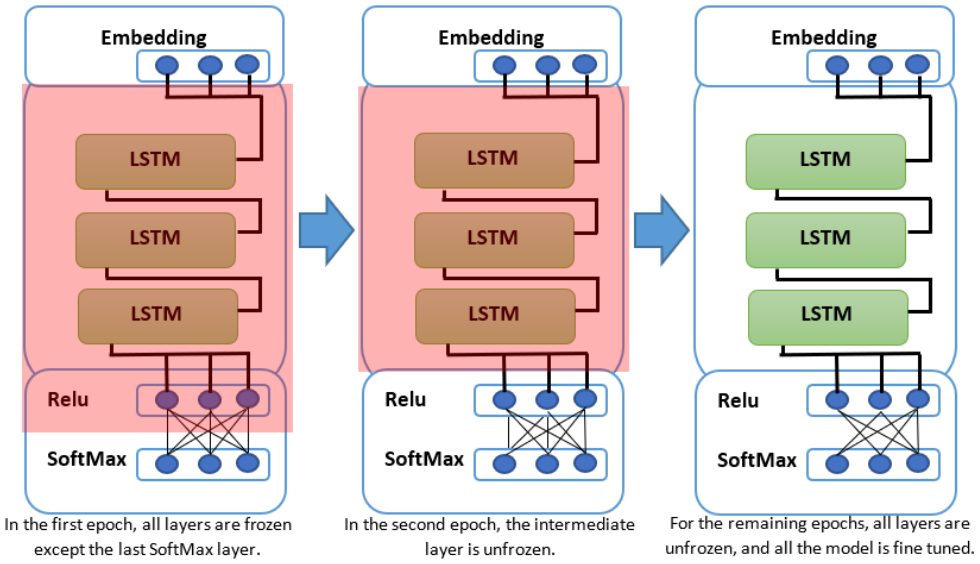

Figure 6: gradual unfreezing the model through epochs.

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. Similar to LSTM [See 2.3.], 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.

4.2.2 Class Diagram

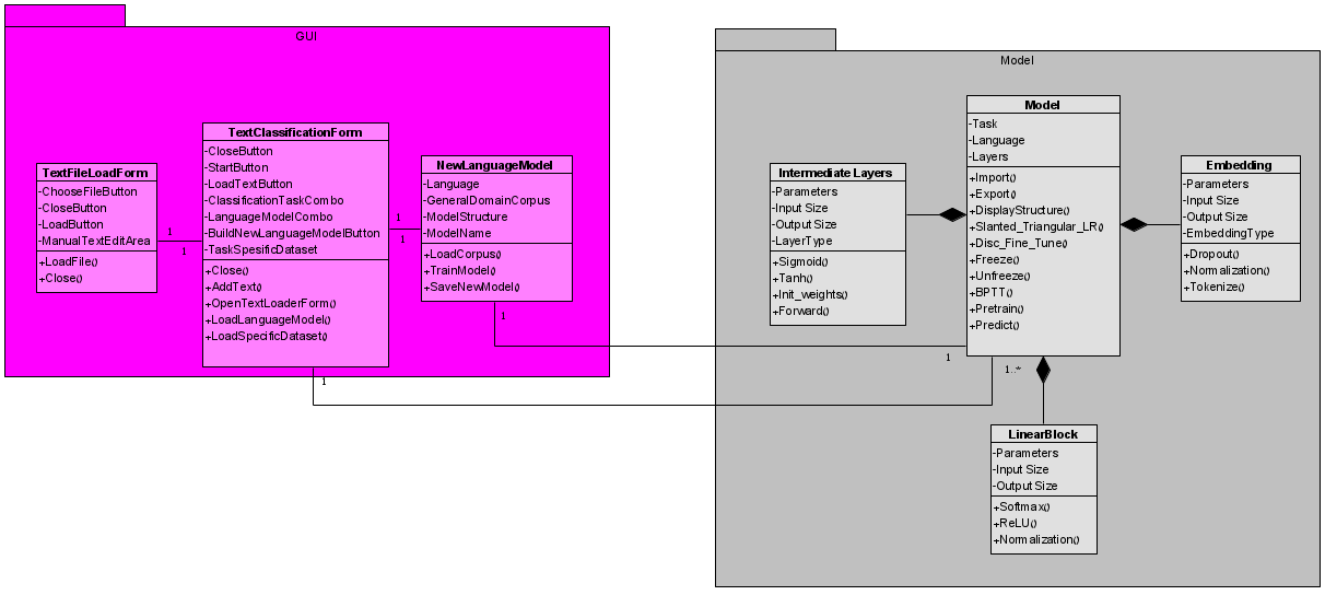

Figure 7: class diagram of the software system

4.2.3 Use Case Diagram

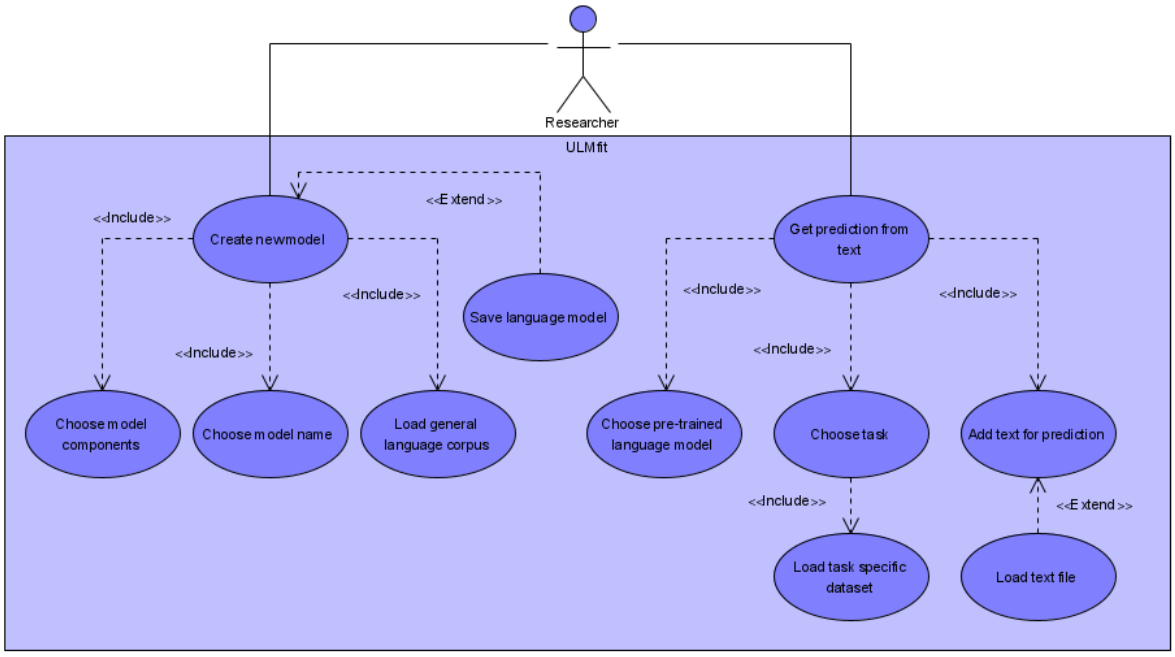

Figure 8: Use case diagram

4.2.4 Graphic User Interface

Our graphic user interface includes 3 forms: text classification form, new language model form and text file load form. The text classification form [see, Fig. 9] is the main window for the application which allows the user to upload a text file, choose an NLP task among the available options and get a prediction as an output from the deep learning model. For further research purposes, the new language model form [see, Fig. 10] allows the user to create a new language model that is used in the ULMFiT algorithm. The user has the possibility to choose the intermediate layers within the model, along with the word embedding layers, the general corpus dataset, and a name for the new model. The text file load form is used for uploading a text file as an input for the language model. Furthermore, the user can input his text manually instead of uploading a text file.

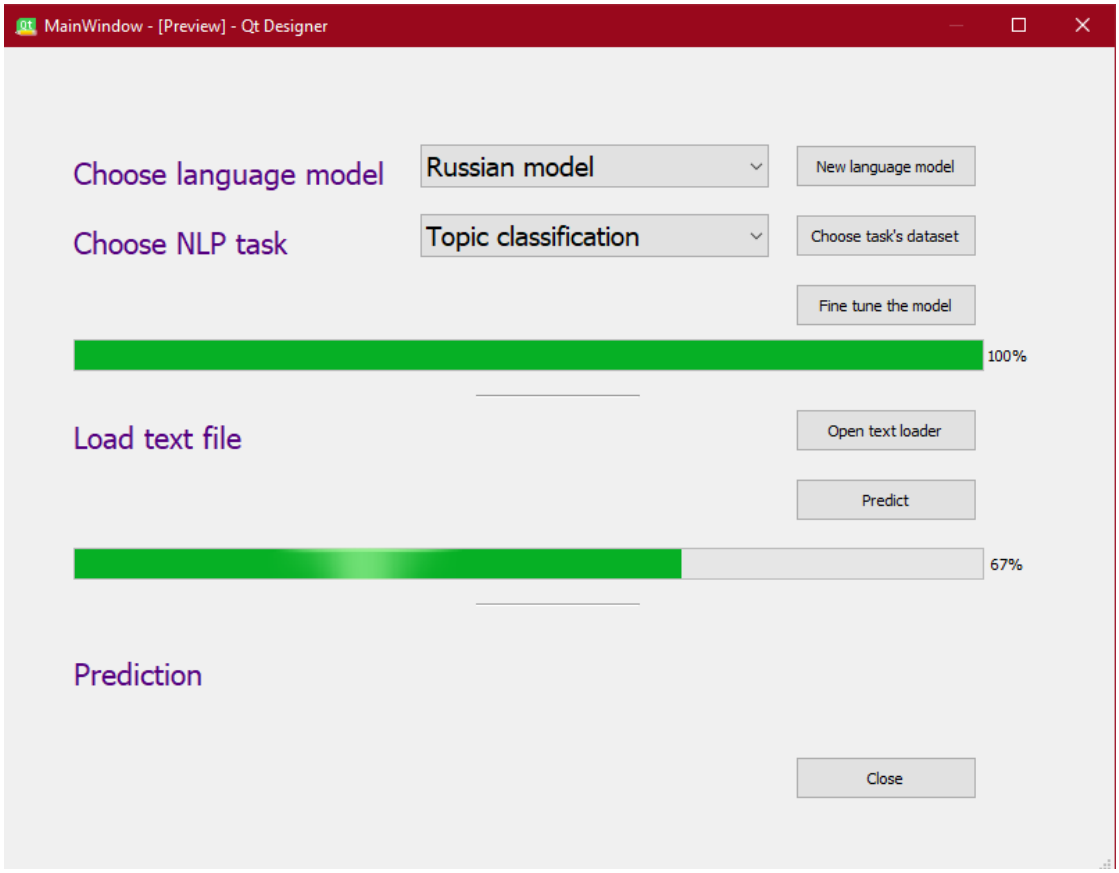

Figure 9: Text classification form

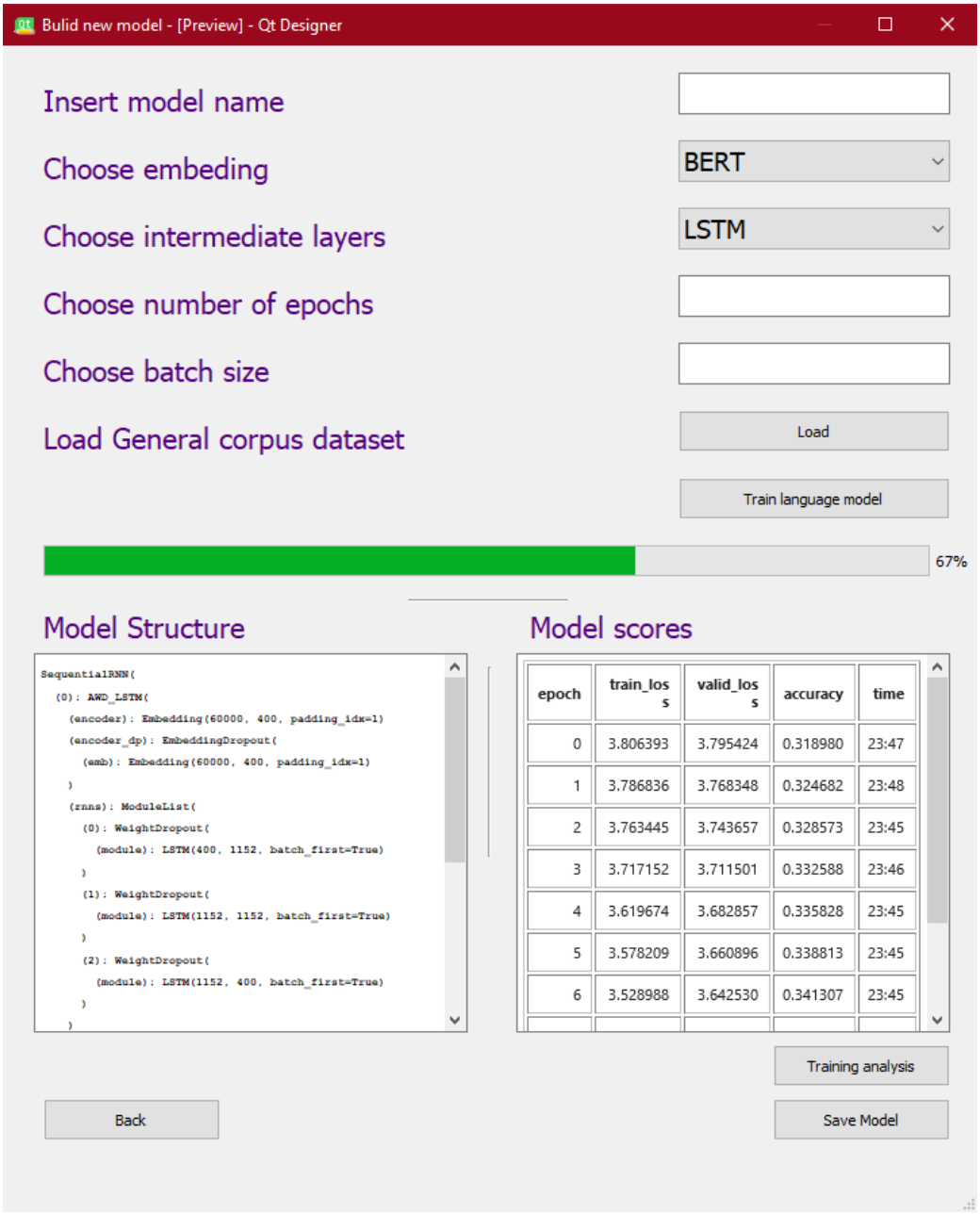

Figure 10: New language model form

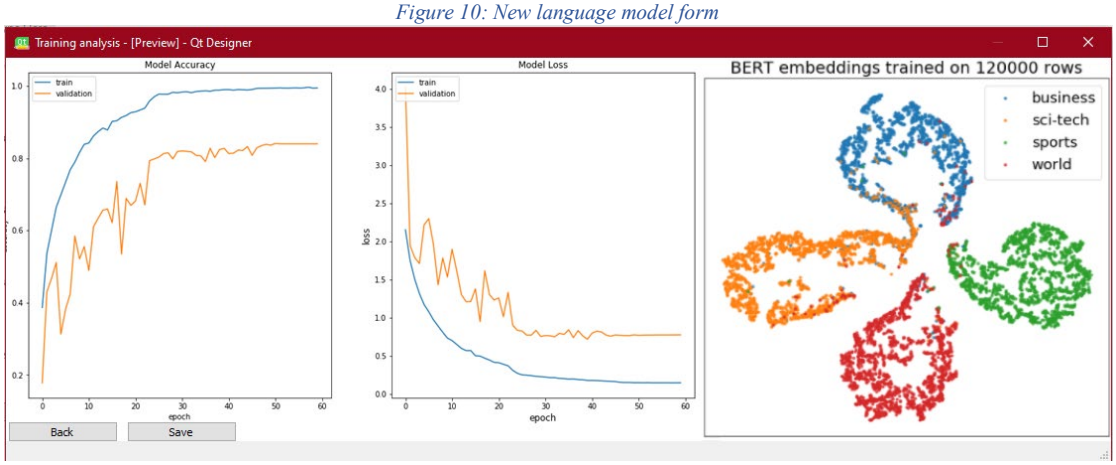

Figure 11: Training analysis form

5. Evaluation/Verification Plan

Our testing is divided into 2 parts: Model testing and GUI testing.

5.1 Model Testing

- **Test if the model can transfer tasks successfully-** when altering between different tasks, check if the prediction is set according to the task in hand.
- **Testing each step of the algorithm:**
  - **General domain language model pretraining –** checking if the model is predicting the next word in a sequence according to the learned language.
  - **Target task language model fine tuning –** checking if the model is predicting the next word in a sequence according to the target task.
  - **Target task classifier –** checking if the loss value is low for the testing dataset.
- **Test the learning process -** run a training step and check if the previous weights are updated.
- **Test the model structure –** print the model's structure and check if it matches the planned model and the model's layers are stacking.
- **There is no overfitting –** check if the loss value for the validation set is much worse than the loss value for the training set.
- **Gradual Unfreezing –** run a training step and verify that the parameters of a frozen layer are unchangeable.

5.2 Graphic User Interface Testing

- **User can upload text file successfully –** check if the text files for the general corpus, task specific dataset and the input text can be uploaded successfully and are useable.
- **User can add manual text –** check if adding a manual text is possible and the text is usable.
- **Model structure display –** check if there is a match between the displayed model structure and the selected model components in the new model form.
- **GUI navigation –** check the links between all forms in the GUI.
- **Forms functionality –** check all widgets functionality.

6. 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.