Faculty of Fundamental Problems of Technology
Wrocław University of Science and Technology

Sentiment Classification using Machine Learning

Paweł Narolski
Nr indeksu: 236685

Thesis Advisor
Piotr Syga, Ph. D.

Politechnika
Wrocławska
Wrocław 2020

Contents

Chapter 1

Introduction

The main goal of this work is to design and implement a document-level text sentiment classification system with a state-of-the-art comparable performance on a practical, real-world domain of movie reviews written in Polish. For this purpose, novel strategies for regularising, pre-training and fine-tuning of recurrent neural networks known as *ULMFiT* are used, combined with inductive transfer learning and subword-based tokenization methods.

1.1 Contents

The following thesis is divided into eight chapters providing a theoretical background on sentiment classification and related machine learning problems, as well as details on the implementation of a proposed sentiment classification system.

Chapter 1 introduces the main goal of this work, methods used to realise that goal and a key measurement of a proposed system's performance.

Chapter 2 defines a problem of document-level sentiment classification, discussing previous work done in the field for Polish language.

Chapter 3 offers a theoretical background on core machine learning and deep learning concepts related to the implemented system.

Chapter 4 builds on the provided theoretical background by exploring key concepts behind recurrent neural networks-based natural language processing, including most notable techniques used in this field.

Chapter 5 presents the reasoning behind the design of the sentiment classification system as well as details on its implementation.

Chapter 6 includes an user manual for the proposed sentiment classification system, as well as the system's platform requirements and dependencies.

Chapter 7 discusses the results of experiments performed on a implemented system, validating the impact of hyperparameters on the overall accuracy of the underlying machine learning model.

Chapter 8 summarises the accomplishments of this work, recapitulating the key insights and performance indicators of an implemented sentiment classification system.

Chapter 2

Document-level sentiment classification

The world-wide-web revolution of early 2000s has brought new ways for people to share their opinions on a variety of subjects at a scale that has never been witnessed before in the world's history.

Being able to *mine* immense amounts of opinionated reviews, articles, blog entries or social media posts from an extensive set of sources, it is crucial to automatically process and determine their underlying sentiment in order to achieve a success in one's field. Such a problem is known as *document-level sentiment classification.*

This chapter presents a background on fundamental concepts behind *sentiment analysis* and *document classification*, providing a reader with a perspective on the document-level sentiment classification problem for documents written in Polish language.

2.1 Sentiment analysis

Sentiment analysis is the computational study of opinions and underlying sentiments expressed in text [42]. The goal of sentiment analysis is to derive information from text about the entity discussed in the opinion, its assessed features (or *aspects*), and the sentiment towards individual aspects of an opinion holder given in an opinion. Formally stated:

Definition 2.1.1. An opinion is a quintuple

$$(e, a, s, h, t), \tag{2.1}$$

where e is the target entity of an opinion, a is the target aspect of the entity e on which the opinion has been given, s is the sentiment of the opinion on the aspect a of entity e, h is the opinion holder and t is the opinion posting time [57].

A sentiment is the underlying feeling, attitude, evaluation or emotion associated with an opinion. Consumer research classifies the sentiment into two distinctive types:

1. *Rational sentiments*, which come from rational reasoning, tangible beliefs, and utilitarian attitudes,
2. *Emotional sentiments*, based on non-tangible and emotional responses to entities that are deeply connected with a person's psychological state of mind [10].

Rational sentiments do not express any emotions and an opinion containing a rational sentiment is also referred to as an *rational opinion.* For example, a sentence *"This car is worth the price"* is a rational opinion. On the other hand, opinions expressing emotional sentiment, such as *"I love my iPhone"* or *"I am bewildered about their service personnel"* are called *emotional emotions* [57].

Sentiments are also classified based on their positive, neutral or negative *orientation* and their *intensity.* A neutral orientation means that a given statement does not contain any sentiment towards the discussed entity (e.g. *"This is my car"*) [57].

Each of the sentiment types and orientations can have different strengths or intensities based on chosen words or phrases (called *sentiment expressions*) and *intensifiers* or *diminishers* used in an opinion. For instance, using word *good* implies lesser intensity of a statement with a positive sentiment than a word *excellent*, and using a diminisher, such as *really* before a word terrible increases the negativity of an expression [57]. Sentiment expressions, intensifiers and diminishers are applied mainly to opinions with positive or negative sentiment orientation.

Unlike factual information, a majority of opinions is subjective. This is caused by each and every single opinion holder coming from a different background, having lived through different experiences, being concerned with different interests, ideologies and beliefs [57]. For instance, two different customers might have bought the same smartphone model, but the first customer has received a defective unit, and the other one - a properly working phone. The first customer might have a negative sentiment towards the phone (or a phone brand) if the fault is discovered, or a positive one, depending on a nature of a fault, the customers' technical knowledge and the overall satisfaction with a unit. This ambiguity poses one of the biggest challenges for the field of sentiment analysis.

2.2 Natural language processing

Natural language processing (or *NLP*) is a subfield of *computer science* and *linguistics* that uses computer-based methods to analyse language in natural text and speech [9]. The goal of NLP is to develop techniques allowing for near-human-level accuracy in tasks such as sentiment analysis, *natural language understanding*, *speech recognition* or *natural language generation*.

Natural Language Processing has relied on the developments in the field of *artificial intelligence* (*AI*), evolving from *symbolic* approaches (based on manually defined sets of rules) to more modern, statistics-based methods for problem-solving [42].

Many recent breakthroughs in NLP brought by *deep learning* have made it possible for automatic language translators and voice assistants, for instance, to become ubiquitous and indispensable parts of our everyday life.

2.2.1 Document definition

In NLP, *document* is an entity containing a distinct text. For instance, a single document might contain the entire contents of a single book, as well as an individual article.

In this work, a term document is used interchangeably with *text document* to emphasise the nature of document's contents.

2.2.2 Text classification

Text classification is a fundamental natural language processing task of assigning a label l for an input text document t from a defined set of labels C known as *classes* based on predefined criteria. It has broad applications in the fields of sentiment analysis, topic labelling and spam detection, among others [100].

For example, having a set of labels $C = \{\mathrm{e-mail}, \mathrm{spam}\}$ and input e-mail subjects t_1 = "WIN LOTTERY, REWARD!" and t_2 = "Report from Research, May 2019" the goal of a text classification of e-mail subject texts is to produce pairs (t_1, l_1) and (t_2, l_2), where l_1 = "spam" and l_2 = "e-mail".

2.3 Document-level sentiment classification

Document-level sentiment classification, also referred to as or *document sentiment analysis* or *document sentiment classification*, is a task of classifying an *opinionated document* as expressing a positive, neutral or negative opinion (or sentiment) [42].

It is assumed in document sentiment analysis that the opinionated text document expresses opinions on a single entity e and contains opinions from a single opinion holder h [57]. Thus, in order for a document to be properly classified using document-level sentiment classification, it should not contain opinions expressed on different entities, or by multiple authors on a single entity.

Considering an example opinionated document containing a review of a book from Amazon.com:

Example 1. "Thrilling read. If you're having trouble sleeping this is perfect for reading just before bed. If you're wide awake and want to learn how to write and publish a scientific paper, it works for that too. I'm a grad student and this book was recommended to me by a prof from my undergrad university and it was worth every penny. It helped me get better at analysing the quality of writing by others [25]."

an accurate document sentiment classification determines the example document to be positive in terms of the opinion holder's h sentiment towards a book entity e.

Document-level sentiment classification task treats sentiment classification as a traditional text classification problem with sentiment orientations as classes [57]. As such, applications of a supervised learning-based solutions are used as a basis for modern document sentiment analysis systems.

2.4 From symbolic to neural networks-based approaches to NLP and document sentiment analysis

Classical approaches to NLP have their roots in *symbolic artificial intelligence*, based on the knowledge of the field of linguistics and involving meticulous analysis of texts using many predefined sets of rules [42].

Modern natural language processing is, contrary to symbolic approaches, heavily based on the latest developments in the area of *deep learning*. Combining classical methods, such as word *tokenization*, with the *deep neural networks*' ability of learning accurate representations of data has allowed for state of the art results in a variety of NLP tasks to be achieved, including text classification, sentiment analysis and machine translation [40, 113, 29, 71, 70].

After an introduction of core machine learning and deep learning concepts in chapter 3, *recurrent neural networks-based* approaches to natural language processing and, specifically, the problem of document sentiment classification are discussed in chapter 4 of this work.

2.5 Polish document sentiment classification

Polish language belongs to a family of highly inflected languages, in which a form or ending of a word depends on it's use in a sentence, reflecting a grammatical tense, case, voice, aspect, person, number, gender, mood, and others [104]. Combining a complex grammar with an extensive set of exceptions, Polish language has been perceived as difficult for automated natural language processing [6, 89, 60].

This section introduces a neural network-based approach to the problem of the Polish document-level sentiment classification, as well as provides an outlook on the previous work done on this subject for Polish opinions.

2.5.1 This work

As the aim of this work is to produce a document-level text sentiment classification system with a state-of-the-art comparable performance, the resulting system is based upon *deep recurrent neural networks*.

The ULMFiT method has been chosen as a foundation of the system, used to produce a final *target classifier* model. Based on promising results from previous works [16, 69], a *sentencepiece*-based tokenizaton method is used on two separate, real-world corpora - *plwiki* and *Filmweb+*, specifically prepared for use in this work.

Compared to previous work (see 2.5.2), our document sentiment classification system is trained on relatively small labelled dataset, Filmweb+, consisting of just 3085 positive and 3085 negative example documents, with an average length of 514 words per document.

Also, the results of this work can be reproduced using commercially available hardware, with the process of pre-training and fine-tuning language models and training target classifier taking less than 10 hours of computing time on `Nvidia GeForce 2080Ti` graphics card.

The motivation between the design and implementation choices made is provided in chapter 5 of this work, and a theoretical background - in chapter 3 and 4.

2.5.2 Previous work

Previous work on the Polish text document sentiment classification has been based on classical NLP approaches [7, 110], *Bayesian* [105, 11, 51] and *support-vector machines*-based [109, 2] classification methods, as well as deep neural networks [16, 50, 95, 69], with the latter producing state-of-the-art results for a Polish language at the time of writing this thesis.

However, a vast majority of the work on document-level sentiment classification has been performed on *corpora* consisting of relatively short texts, such as posts from Twitter [69, 50, 11], short user rating comments from Filmweb [95] and Opineo store and product reviews [2]. In fact, no recent work on Polish document sentiment analysis that involved use of a dataset consisting of long (over 1000 characters on average), real-world documents.

Some of the neural networks-based approaches to document-level sentiment classification have leveraged the *pre-training* and *fine-tuning* of *language models* as well as *inductive transfer learning* [16, 69, 50, 95], using methods known *Universal Language Model Fine-Tuning* (or *ULMFiT*) [40] and *Bidirectional Encoder Representations from Transformers* (or *BERT*) [20]. Both of these methods have produced satisfactory results.

Chapter 3

Background on machine learning

The following chapter introduces the core problems of machine learning and deep learning involved with a recurrent neural networks-based natural language processing systems used to perform document-level sentiment classification.

It defines core concepts, such as *transfer learning*, *regularisation*, *backpropagation* and *recurrent neural networks* referred to in the latter parts of this work.

3.1 Machine learning

Machine learning (or *ML*) is a subfield of *artificial intelligence* that gives computers the ability to learn without being explicitly programmed [83].

A formal definition of *learning* is provided by Tom Mitchell [64]:

Definition 3.1.1. A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

To put a definition 3.1.1 in layman's terms, a computer program learns when it is able to interpret new data (experiences E) in such a way, that it is able to solve a given problem (task T) with a better outcome (as measured by P).

Machine learning-based classification systems, for instance, learn how to accurately label a provided input by being presented with a set of example inputs for each category called the *training set.* For such systems:

1. The task T is to classify an input based on its distinctive features,
2. The experience E, from which a computer program learns, is a *sample* from a training set,
3. The performance measure P can be defined in numerous ways deemed appropriate; commonly, ratio of correctly classified inputs - a metric called *accuracy* - is used.

Sentiment classification system is an example of such a system, for which the task T is to classify an opinionated document based on its contents and an experience E is an individual, already labelled document. A P can be measured by an accuracy, *precision* or *F1 score* metrics, among others.

A core part of the machine learning system is called a machine learning *model*, which is an entity that has obtained the knowledge from all the experiences $E_i \in \{E_1, \ldots, E_n\}$ belonging to a training set. The process of presenting a machine learning system with new examples from a training set is called *training.*

Machine learning is a broad field in its own, with many subfields, most notable of which include *representation learning* and *deep learning.* A relation between artificial intelligence, machine learning and its subfields is shown in figure 3.1.

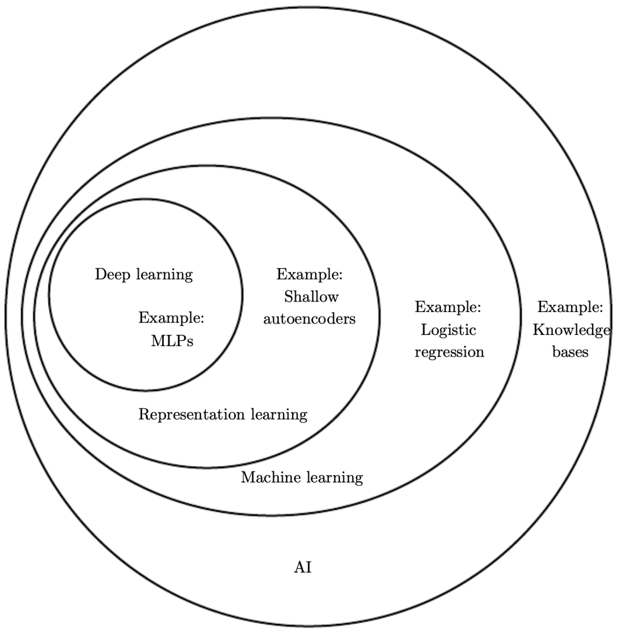

Figure 3.1: A diagram illustrating the relationships between the fields of artificial intelligence, machine learning, representation learning and deep learning [29].

3.1.1 Types of machine learning systems

Machine learning systems are classified based on the human input needed during the training process into:

1. *Supervised learning* systems, which learn how to interpret new data given some input data with associated, desired outputs (or *labels*),
2. *Semi-supervised learning* systems, which learn from input data for which only some labels are available,
3. *Unsupervised learning* systems, which learn the correlations between given *unlabelled* input data.

3.1.2 Inductive and transductive learning

Transduction, in the context of statistical learning, refers to predicting specific results of drawing from a given set of examples. A classical example of a transductive algorithm is the *K-Nearest Neighbours* algorithm, which uses the training data directly to make a prediction.

Contrary to transduction, *induction* in machine learning is a process of deducing general rules from observed training cases.

3.1.3 Representation learning

The performance of machine learning systems depends heavily on a *representation of data* used [4].

An example ML system based on a *logistic regression* algorithm [65] used to recommend a cesarean delivery produces its output based on an input vector of features, representing neonatal birth weight or gestational age at birth, among others. Such a system requires its operator to manually *pre-process* input data not only for the purposes of training, but also - in this case - in order to produce a valid classification.

In general, much of the actual effort involved in deploying a machine learning algorithm has involved the design process of task-oriented pre-processing pipelines and data transformations, resulting in a representation of data that allows for accurate outputs, as illustrated on a figure 3.2 [4].

However, many problems cannot be solved through a manual engineering of data representations. Automated face detection, for example, was an unsolvable problem until as recently as the year 2001 [93, 67],

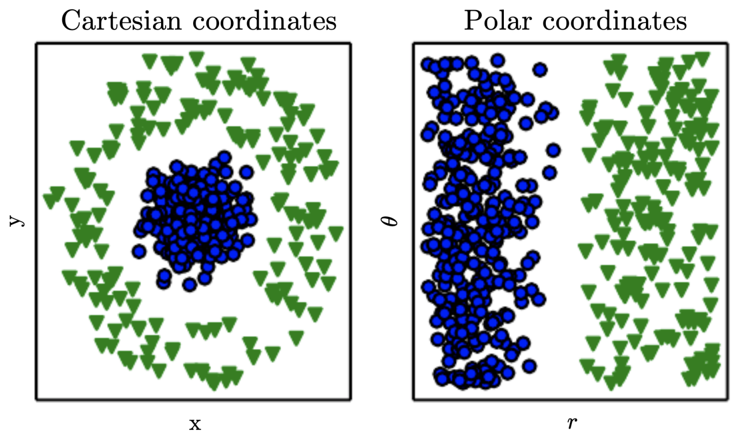

Figure 3.2: Using a different representation of data may result in a much simpler solution for a target machine learning task. In this example, a change of coordinate system from Cartesian to polar allows to perform a classification of objects belonging to class of dots and triangles by separating them with just a single line [29].

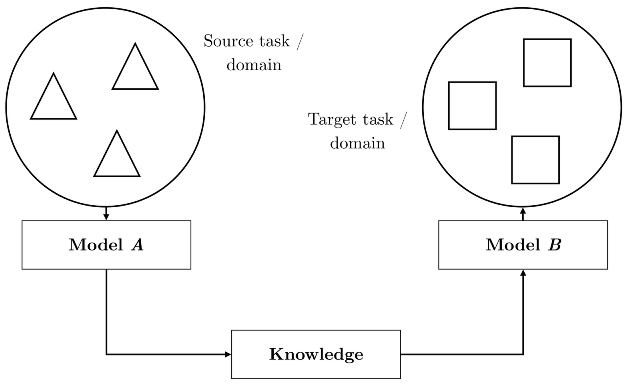

Figure 3.3: Transfer learning allows for the use of knowledge obtained by model A when solving a source task (from a source domain) in order to improve model B's accuracy on a target task (from a target domain) [82].

when an integral image representation was first proposed.

This difficulty has lead to an emergence of *representation learning*, a set of machine learning techniques which allow a system to automatically discover the representations needed for a given task and replace a tedious, aforementioned process called *feature engineering* [101].

Current proliferation of object detection, speech recognition or NLP systems [4], among others, is attributed to the emergence of effective representation learning methods, such as *deep learning*.

3.1.4 Transfer learning

Transfer learning refers to a machine learning algorithm's ability of exploiting commonalities between given learning tasks [4] through an use of knowledge obtained in one learning task in order to improve a system's performance in another task.

Inductive transfer learning

Inductive transfer learning is a method of improving a generalisation's accuracy on a target task's domain with limited labelled data available through the use of labelled data of related source domain [85].

The assumption behind inductive transfer learning is that it might be easy to obtain large quantity of quality data from a broader domain, which will help the machine learning system "understand" complex

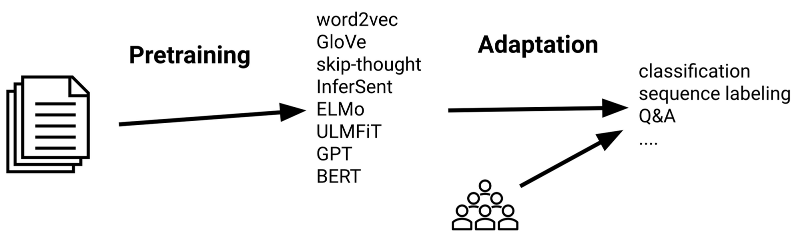

Figure 3.4: In sequential transfer learning a machine learning model is first trained on a large set of unlabelled training data and then used to improve an accuracy on the target task, such as the document classification [82].

relations between certain features of data and then use them to produce a model capable of solving a subproblem which smaller amount of data is available for. This process is also referred to as *pre-training*.

Inductive transfer learning has helped to improve accuracy on many computer vision tasks, such as image segmentation, classification, or object detection, where it common to fine-tune models pre-trained on datasets such as ImageNet or MS-COCO [40].

In NLP, inductive transfer learning had been thought to be applicable only for semantically-similar tasks [66] until 2018, when *Universal Language Model Fine-tuning* (*ULMFiT*) method [40] using*sequential transfer learning* was proposed, perceived today as an idea that has lead to the most improvements in the field so far [82].

Sequential transfer learning

Sequential transfer learning is a method that has allowed to significantly increase accuracy in when inductive transfer learning is applied in the field of Natural Language Processing.

It is applied by pre-training data representations on a large, unlabelled text *corpus* (set of documents) and then adapting these representations for a target, supervised task, such as classification, Q&A or sequence labelling [82].

3.1.5 Evaluating machine learning models

To evaluate the accuracy of a machine learning model, it is required to define:

1. A performance metric P, which quantifies the accuracy between predicted and true values,
2. A set of observations outside the training data, whose predictions are tested by the metric [91].

In order for a model to be applicable in a real-word scenario, it is required to *generalise* well, meaning that it is able to perform well (according to P) on input data not available during the process of training. Evaluation of model's accuracy and generalisation ability can be performed through *hold out* method.

Hold out method

In a hold out method, a training dataset is randomly divided into three sets:

1. A training set, used to train a given model, which contains 70% of available data,
2. A *validation* set, used to measure the impact of *hyperparameters* change on the performance of the model, which contains 10-15% of remaining data,
3. A *test* set, used to estimate a generalisation of a model, which also contains 10-15% of remaining data [91].

An usage of a hold out method enables us to verify the performance of a model on two separate sets, the validation set and a test set, which reduces the risk of *overfitting* a model to a validation set [26].

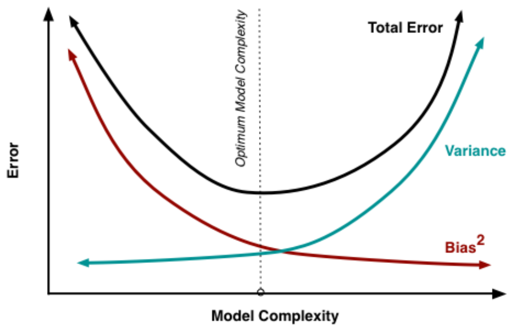

Figure 3.5: An illustration of bias-variance trade-off. A reduction of bias results in an increase of variance of the model, and vice versa. An optimal model complexity is achieved when an increase of bias leads to decrease of variance of a model [23].

Bias-variance trade-off

Assuming that a machine learning model can be trained on different training sets and evaluated more than once, we define *bias* as as the difference between the expected (or average) predictions of a resulting realisations of a model and the target output. Models with high bias value tend to be overtly simplistic, leading to low accuracy on training, validation and test sets.

Similarly, a *variance* of a model is a measurement of how the predictions for a fixed input vary between different realisations of a machine learning model. High variance results in a complex model, which is able to correctly describe relations between data from a training set, but unable to generalise well to new inputs [23].

A *bias-variance trade-off* is a central problem in a machine learning of finding a balance between capturing the regularities in the training data and a model's ability to generalise well to unseen data [99].

In order to formally define the impact of bias and variance on the predictions of the model, let's denote Y as the variable a machine learning model is trying to predict, and X as our inputs to the model. Then, we may assume that X's relation to Y can be defined as:

$$Y = f(X) + \epsilon, \tag{3.1}$$

where the error term ϵ is normally distributed with a mean of zero, i.e. $e \sim \mathcal{N}(0, \sigma_\epsilon)$.

Since a model $\hat{f}(X)$ of $f(X)$ can be estimated using a machine model of choice, an expected square prediction error at point x is:

$$Err(x) = \mathbb{E}[(Y - \hat{f}(x))^2], \tag{3.2}$$

which, decomposed, has bias and variance components:

$$Err(x) = \underbrace{(\mathbb{E}[\hat{f}(x)] - f(x))^2}_{\text{Squared bias}} + \underbrace{\mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2]}_{\text{Variance}} + \underbrace{\sigma_\epsilon^2}_{E_{irr}}, \tag{3.3}$$

where E_{irr} is an irreducible error resulting from a noise in a defined relation that cannot be reduced by any model [34, 23].

In conclusion, finding a good balance between bias and variance results in an improved accuracy of a machine learning model.

Overfitting and underfitting

Overfitting occurs when a machine learning model performs well on a training data, but fails to generalise well to new, previously unseen data. An overfitting model shows low *bias* and high *variance*.

An *underfitting* model, on the other hand, is a model that is unable to produce accurate results on training data (and new data, as well), most often due to its insufficient complexity, reflected in low variance and high bias of a model [8].

Convergence

In mathematics, a sequence S_n is said to be convergent to a given limit L, $\lim_{n\to\infty} S_n = L$ if, $\forall\epsilon > 0$ there exists N such, that $|S_n - S| < \epsilon$ for $n > N$. In machine learning, it is often said that a model *converges*, meaning that an error rate of an algorithm after each training iteration approaches a global (or local) minimum [97].

3.1.6 Gradient descent

Gradient descent is an optimisation algorithm used to minimise a value of a given objective function $f(\theta)$, where $\theta \in \mathbb{R}^d$ are the machine learning model's parameters, by iteratively moving in a direction of the steepest descent of $f(\theta)$ as defined by the negative of the gradient [30].

Learning rate

A parameter η called the *learning rate* is defined for the purposes of gradient descent algorithm, which defines the size of a step taken to reach a local minimum of a given objective function $f(\theta)$.

Optimisation

Optimisation refers to a task of either minimising or maximising a value of a given function $f(x)$ by altering x. Most optimisation problems involve the minimisation of $f(x)$, since the maximisation may be accomplished via a minimisation algorithm by minimising $-f(x)$ [29].

Gradient vector

Gradient of a scalar-valued multi-variable function $f(x_1, x_2, \dots)$, denoted ∇f, packages all its partial derivative information into a vector [14]:

$$\nabla f = \begin{bmatrix} \dfrac{\delta f}{\delta x_1} \\ \dfrac{\delta f}{\delta x_2} \\ \vdots \end{bmatrix}. \tag{3.4}$$

If at a given point $p \in f(x_1, x_2, \dots)$ the gradient of a function is not a zero vector, the direction of ∇f is the direction of the fastest increase of the function f at p. The magnitude of ∇f is the rate of increase in that direction [1, 102].

Batch gradient descent

A *batch gradient descent* method computes the gradient ∇ of the cost function C with respect to parameters θ of the entire training set [79]:

$$\theta = \theta - \eta \cdot \nabla_\theta C(\theta) \tag{3.5}$$

Computing the gradient for an entire training set at a time is inpractival, however, due to extensive computations required and memory requirements posed by larger datasets.

Stochastic gradient descent

Stochastic gradient descent method performs an update of θ for each training example x_i and output y_i [79]:

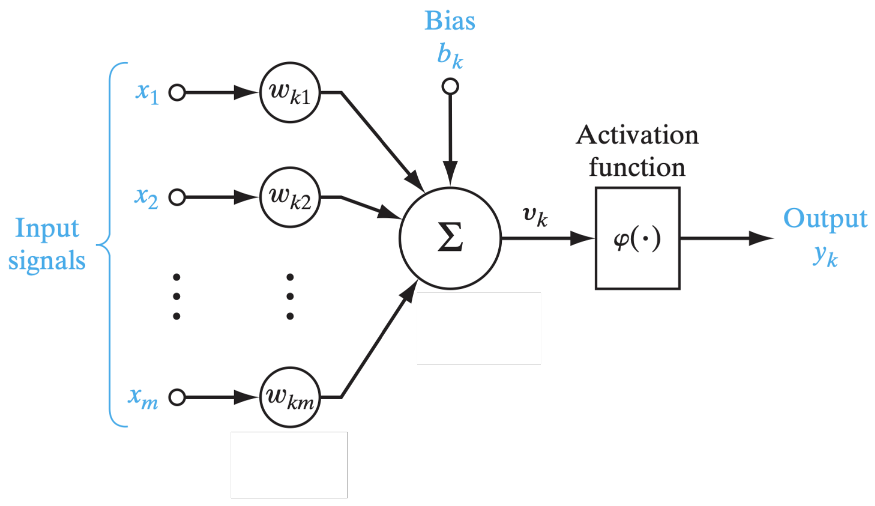

Figure 3.6: A diagram illustrating the input data flow in an artificial neuron [35].

$$\theta = \theta - \eta \cdot \nabla_\theta C(\theta; x_i, y_i) \tag{3.6}$$

Unlike the batch gradient descent method, stochastic gradient descent method computes gradients for distinctive members from a training set only.

Mini-batch gradient descent

Mini-batch gradient descent builds on the ideas of stochastic gradient descent and batch gradient descent, performing an update for every mini-batch of n training examples:

$$\theta = \theta - \eta \cdot \nabla_\theta C(\theta; \boldsymbol{x}^{(i:i+n)}, \boldsymbol{y}^{(i:i+n)}), \tag{3.7}$$

where $\boldsymbol{x}^{(i:i+n)}, \boldsymbol{y}^{(i:i+n)}$ denote vectors of elements from index i to index $i+n$ taken from vector $\boldsymbol{x}$ of all training examples and vector $\boldsymbol{y}$ of all outputs respectively.

An application of mini-batch gradient descent reduces the variance of the parameter updates, which is introduced by stochastic gradient descent, leading to a more stable convergence. It can also benefit greatly from matrix-based calculation optimisations, making it very efficient [79].

3.2 Neural networks

Neural networks, or artificial neural networks (ANN) are machine learning models consisting of interconnected *artificial neurons* gradually processing input data through pre-learned transformations in order to produce an accurate output for a given task.

ANNs have been inspired by the biological neural networks which constitute animal brains in recognition of their superior ability to perform certain computations, such as pattern recognition or perception [35] instantaneously, as opposed to traditional, rule-based computing systems.

Today, neural network models are the heart of cutting-edge developments in the field of artificial intelligence, exceeding the human performance in tasks, such as a game of *Go* [87].

3.2.1 Artificial neurons

An *artificial neuron* (also referred to as neuron or perceptron) is an information-processing unit fundamental to the operation of a neural network [35].

A neuron N_k is defined by a set of inputs $x_1, x_2, \ldots, x_n$, a set of weights $w_1, w_2, \ldots, w_n$, where each weight w_i corresponds to an input x_i, a bias value b_k, an activation function $\phi(\cdot)$ and an output y_k.

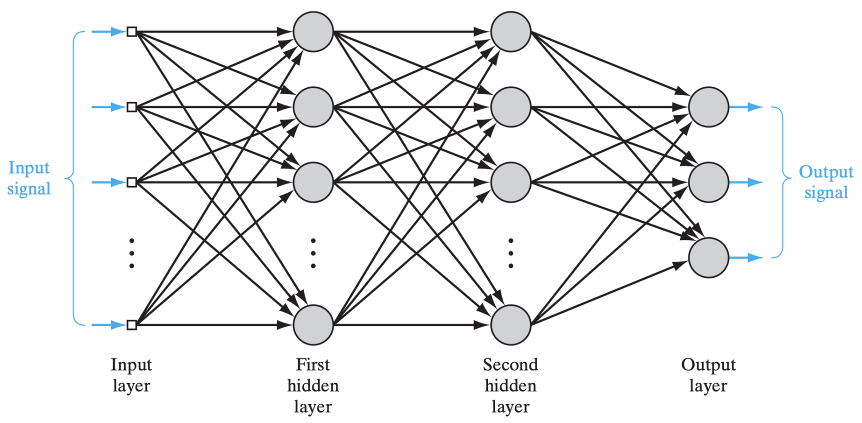

Figure 3.7: An example of a multi-layer perceptron neural network consisting of an input layer, two hidden layers and an output layer. [35].

Neuron's output y_k is calculated by applying an activation function ϕ on top of a weighted sum of inputs, to which a bias value b_k is added:

$$y_k = \phi(\sum_{i=1}^{n}(x_i w_i) + b_k). \tag{3.8}$$

The bias value b_k is used to affect the value an activation function of a N_k neuron outputs. In another, often used notation, b_k is applied as a weight for an input x_0, which has a fixed value of 1. This simplifies the equation 3.8:

$$y_k = \phi(\sum_{i=0}^{n} x_i w_i). \tag{3.9}$$

3.2.2 Multi-layer perceptron

A *multi-layer perceptron* (or *MLP*) is a quintessential neural network model in which multiple neurons are grouped into *layers* [29]. It is a *feedforward neural network*, which means that the connections between the neurons do not form a cycle.

A MLP consists of an *input layer*, followed by one or more *hidden layers* and an final layer called the *output layer*, producing the output of the network. The overall structure of a neural network is also referred to as an *architecture* of a network.

The goal of a multi-layer perceptron is to approximate a function $f^\star$. For instance, in classification tasks, where $y = f^\star(x)$ performs a mapping from an input x to a category y, a MLP defines a mapping $y = f(x, \theta)$ and learns θ, the value of parameters resulting in the best $f^\star$ approximation [29].

3.2.3 Hidden layers

Hidden layers are all the layers in a neural network except the input layer and the output layer of the network.

While it has been thought that a multi-layer perceptron with just one hidden layer consisting of a large enough set of neurons can model even the most complex mathematical functions [26], deep neural networks have a much higher *parameter efficiency* compared to the shallow neural networks. This means that neural networks with multiple hidden layers are able to model complex functions using exponentially fewer neurons, which results in a faster-to-train network.

Deploying multiple hidden layers also results in a network which can learn appropriate data representations gradually: first, modelling low-level structures, then gradually combining them into more complex structures to produce a final, high-level representation used to produce a final output of a network.

Such an approach can be thought as an application of the *divide-and-conquer* strategy in the context of machine learning. It also yields a notable advantage in the resulting neural network being able to

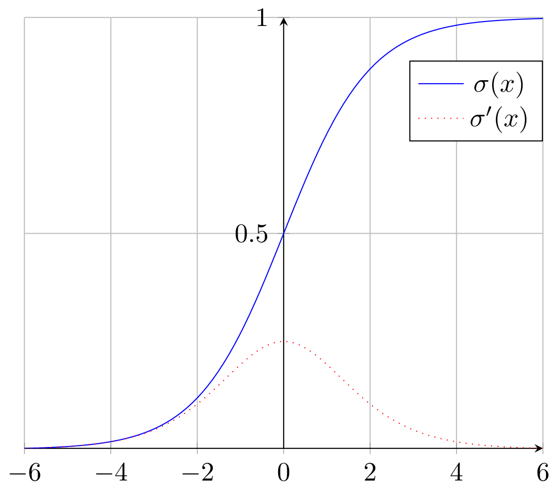

Figure 3.8: A plot of a sigmoid activation function $\sigma(x)$ and its derivative $\sigma'(x)$.

generalise to new datasets through an application of transfer learning - lower layers of deep neural network can be reused for the purpose of training similar network on new sets of input and output data, instead of performing a random initialisation of weights and biases for these layers [26].

3.2.4 Activation functions

Activation functions are functions applied on the activation value of an artificial neuron in a neural network in order to produce a final output of a neuron. Their application produces an uniform mapping from activation values to outputs of a neural network's layer.

An activation function should be a function that is monotonic, in order for it to produce outputs preserving relationships between pairs of input data, and differentiable [96].

Sigmoidal activation functions

A *sigmoid function*, also referred to as sigmoid curve or logistic function (see figure 3.8) is a function defined by the formula:

$$\sigma(x) = \frac{1}{1 - e^{-x}}. \tag{3.10}$$

While the sigmoid function was one of the first activation functions used in neural networks, its modern applications are limited due to a *vanishing gradient* problem. For very high or very low values of $\sigma(x)$, its derivative's $\sigma'(x)$ values are small, causing equally marginal change to the prediction of a model. Computing a value of sigmoid function also requires significantly more processing power as opposed to the $ReLU$ function.

Rectified Linear Activation Unit

Rectified Linear Activation Unit (or ReLU) is an activation function defined as:

$$f(x) = max(0, x) = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{3.11}$$

It is one of the most commonly used activation functions in modern deep learning models [3] due to the fact that the value of ReLU function can be easily computed. An *ImageNet* image classification neural network has been shown to converge over 25% faster when ReLU was used instead of *tanh* function, for instance [52].

Being a differentiable function, ReLU allows the *backpropagation* algorithm to be used during the process of training a neural network.

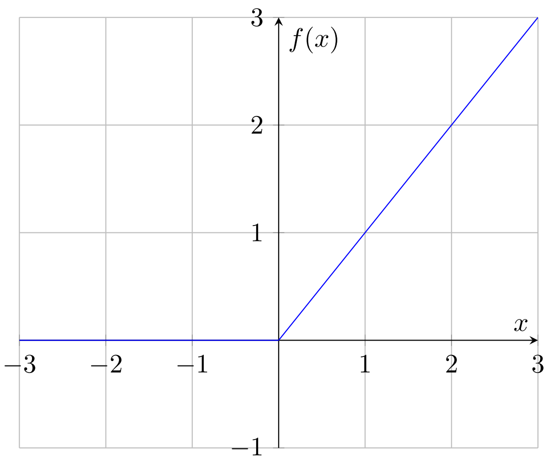

Figure 3.9: A Rectified Linear Activation Unit is one of the most often used activation functions in modern neural networks.

Softmax

Softmax function that takes as an input a vector of K real numbers and normalises it into a probability distribution consisting of K probabilities proportional to the exponential of the input numbers [108]. It is defined as:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K, \tag{3.12}$$

where on each element z_i of an input vector $\mathbf{z}$ a standard exponential function is applied and the calculated value is normalised through a division by the sum of all the exponents, ensuring that the sum of output vector's components $\sum_{i=1}^{K} \sigma(\mathbf{z})_i$ is equal to 1 and all the components of $\sigma(\mathbf{z})$ are in the range of $(0, 1)$. What's more, a relation between the value of an input and output probability is maintained - larger input components correspond to larger probabilities, and so forth.

Softmax is used as an activation function in neural networks to convert the outputs of neurons from a given layer into probability values, corresponding for instance to model's certainty that the input belongs to a given class in a classification problem.

3.2.5 Hyperparameters

Hyperparameters in the context of neural networks are all the parameters of a given network which value is set before the beginning of a learning process.

They can be classified as model hyperparameters, which, in the context of neural networks include a number of hidden layers in a neural network, an amount of neurons per hidden layer and activation functions used in neurons, or algorithm hyperparameters, which affect the speed and outcomes of the learning process, such as the learning rate or mini-batch size [103].

3.2.6 Backpropagation

Backpropagation method is an algorithm used during the training process of a neural network to calculate the changes of neuron weights and biases needed to be applied in order to minimise a given cost function, and thus improve an accuracy of the model.

The method's efficiency has allowed an application of neural networks to a much wider field of problems, that were previously off-limits due to time and cost constraints, due to its ability to produce internal representations of data on par with those manually-engineered, or even those found in biological neural networks [46].

Notation

We use $w^l_{j,k}$ to denote the weight for the connection from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer.

For the network biases, we denote b^l_j as the bias of the i^{th} neuron in the l^{th} layer of the network, and for activations, we use a^l_j for the activations of the j^{th} neuron in the l^{th} layer.

The activation a^l_j of the j^{th} neuron in the network is related to the activations in the $(l-1)^{th}$ layer by the equation

$$a^l_j = \sigma(\sum_k w^l_{j,k} a^{l-1}_k + b^l_j), \tag{3.13}$$

where the sum is over all neurons k in the $(l-1)^{th}$ layer.

In order to rewrite the expression 3.13 in a matrix form, the *weight matrix* $\boldsymbol{w^l}$ is defined for each layer l, which is populated by weights connecting to the l^{th} layer of neurons, which means, that the entry in the j^{th} row and k^{th} column is $w^l_{j,k}$. Similarly, for each layer l a bias vector $\boldsymbol{b^l}$ is defined, which components are defined as b^l_j values.

Having these matrices defined, the equation 3.13 can be rewritten as

$$a^l = \sigma(w^l a^{l-1} + b^l), \tag{3.14}$$

which means, that in order to calculate the activations of neurons in one layer, all we have to do is to apply the weight matrix to the activations from a previous layer, then add the bias vector, and, finally, apply the σ activation function. We also define

$$\boldsymbol{z^l} \equiv w^l a^{l-1} + b^l \tag{3.15}$$

where $\boldsymbol{z^l}$ is called the *weighted input* to the neurons in layer l [68].

Cost function assumptions

In order for the backpropagation method to be applied, we need to assume that our cost function C can be expressed as an average $C = \frac{1}{n}\sum_x C_x$ over cost functions C_x,where x is an individual training example. This is due to the fact that backpropagation allows us to compute partial derivatives $\frac{\delta C_x}{\delta w}$ and $\frac{\delta C_x}{\delta b}$, which $\frac{\delta C}{\delta w}$ and $\frac{\delta C}{\delta b}$ are recovered from by averaging over training examples. We will also assume for now that the training example x is fixed, and refer to the cost C_x as C, until otherwise noted.

We also need to make an assumption that a cost function can be expressed as a function of the outputs from a neural network $C = C(\boldsymbol{a^L})$. For example, a quadratic cost function $C = \frac{1}{2n}\sum_x ||y(x) - \boldsymbol{a^L}(x)||^2$ for a single training example x may be written as :

$$C = \frac{1}{2}||y - \boldsymbol{a^L}||^2 = \frac{1}{2}\sum_j (y_j - a^L_j)^2, \tag{3.16}$$

and thus is a function of the output activations. While the function also depends on the desired output y, it's important to remember that both the training example x and y are fixed, and as such it is sound to regard C as a function of $\boldsymbol{a^L}$, where y is merely a parameter helping to define C [68].

Hadamard product

Let's suppose that $\boldsymbol{s}$ and $\boldsymbol{t}$ are two vectors of the same dimension. An *element-wise* product of two vectors, also known as the *Hadamard product*, is denoted as $\boldsymbol{s} \odot \boldsymbol{t}$, where:

$$(\boldsymbol{s} \odot \boldsymbol{t})_j = s_j t_j \tag{3.17}$$

Backpropagation algorithm

The backpropagation algorithm provides a way to compute a gradient of the cost function:

1. **Input** x: set the corresponding activation a^1 for the input layer
2. **Feedforward**: for each $l = 2, 3, \dots, L$ compute $\boldsymbol{z^L} = \boldsymbol{w^l a^{l-1}} + \boldsymbol{b^l}$ and $\boldsymbol{a^l} = \sigma(\boldsymbol{z^l})$
3. **Output error $\boldsymbol{\delta^L}$**: compute the vector $\boldsymbol{d^L} = \Delta_a C \odot \delta'(\boldsymbol{z^L})$
4. **Backpropagation of the error**: for each $l = L-1, L-2 \dots, 2$compute $\boldsymbol{\delta^l} = ((\boldsymbol{w^{l+1}})^T \delta^{l+1}) \odot \sigma'(\boldsymbol{z^l})$
5. **Output**: the gradient of the cost function, given by $\frac{\delta C}{\delta w^l_{j,k}} = \boldsymbol{a_k^{l-1} \delta_j^l}$ and $\frac{\delta C}{\delta b^l_j} = \boldsymbol{\delta_j^l}$

An error vector $\boldsymbol{\delta^j}$ is computed starting from the final layer, which is a consequence of the fact that the cost is a function of outputs from the network. The chain rule is applied repeatedly to evaluate the cost variations with weights and biases from earlier layers in the network [68].

3.2.7 Training neural networks

In order to train a neural network, the backpropagation method is combined with a learning algorithm, such as the mini-batch gradient descent, in order to compute a gradient of the cost function for many training examples.

Mini-batch gradient descent-based backpropagation algorithm

Given a mini-batch of m training examples:

1. For each training example x: set the corresponding activation $a^{x,1}$ and perform:
   - **Feed-forward**: for each $l = 2, 3, \dots, L$ compute $\boldsymbol{z^{x,L}} = \boldsymbol{w^l a^{x,l-1}} + \boldsymbol{b^l}$ and $\boldsymbol{a^{x,l}} = \sigma(\boldsymbol{z^{x,l}})$
   - **Output error $\boldsymbol{\delta^{x,L}}$**: compute the vector $\boldsymbol{d^{x,L}} = \Delta_a C_x \odot \delta'(\boldsymbol{z^{x,L}})$
   - **Backpropagation of the error**:
     for each $l = L-1, L-2 \dots, 2$compute $\boldsymbol{\delta^{x,l}} = ((\boldsymbol{w^{l+1}})^T \delta^{x,l+1}) \odot \sigma'(\boldsymbol{z^{x,l}})$
2. **Gradient descent**: for each $l = L, L-1, \dots, 2$ update
   - weights, according to the rule $w^l \to w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$
   - biases, according to $b^l \to b^l - \frac{\nabla}{m} \sum_x \delta^{x,l}$.

An implementation of stochastic gradient descent will also require two outer loops, one generating mini-batches of m training examples and another iterating over multiple epochs of training, which were omitted for simplicity [68].

3.2.8 Regularisation

In order to reduce overfitting in neural networks, a set of methods known as *regularisation* techniques has been developed in order improve the generalisation from the training data to the test data. An example of such an technique is a *L2 regularisation* method [68].

L2 regularisation

Weight decay, also known as the L2 regularisation, is one of the more commonly used regularisation techniques, which works by adding an extra term to the cost function C - the sum of all squared weights in the neural network:

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2, \tag{3.18}$$

where C_0 is the unmodified cost function, $\lambda > 0$ is an *regularisation parameter*, and n - the size of an training set. A quadratic cost function with L2 regularisation applied is defined then as follows:

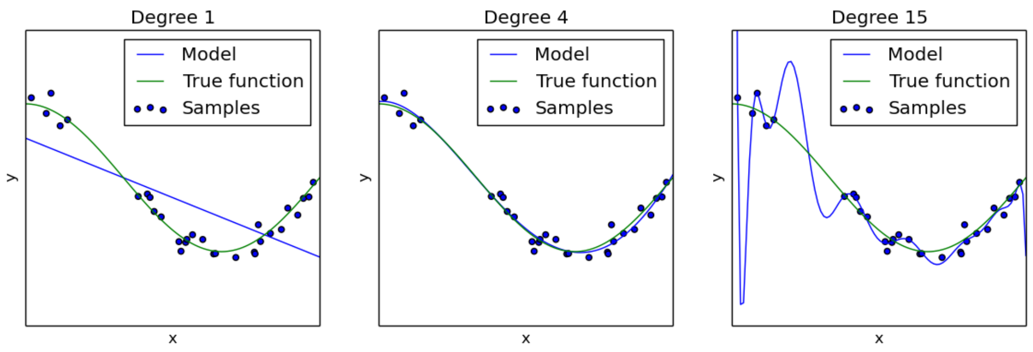

Figure 3.10: Regularisation helps trained models to generalise better to new data by limiting the complexity of the model caused by an overfitting to the noise in the training data. An example of underfitting, a good bias-variance trade-off and overfitting is shown here accordingly in plots depicting a degree 1-, 4- and 15- polynomial attempting to match the training samples given [19].

$$C = \frac{1}{2n}\sum_x ||y - a^L||^2 + \frac{\lambda}{2n}\sum_w w^2, \tag{3.19}$$

For small values of λ, the regularisation has an effect of of minimising the original cost function, and for the large values of λ - affects the values of the network's weights, making it preferred to learn smaller values [68].

Dropout

Arguably the most popular regularisation technique for neural networks is known as *dropout*. Introduced in 2012, it has proven to improve the accuracy of state-of-the-art models by 1-2%, decreasing their error rate by almost 40% [26].

Dropout works by assigning each neuron in the neural network excluding an output layer a new hyperparameter, which is the probability p (most often set to $p = \frac{1}{2}$) of a neuron being omitted during a single training step. After the training, the neurons are no longer excluded from the computations.

Applying such an omission during training has been shown to produce subsequent data representations by neurons which contribute towards the solution of a given task, instead of representations helpful only in context of several other feature detectors. This has an effect of training multiple training networks with different hyperparameters and then averaging their output to produce a final output - but is much more computationally efficient [38].

3.3 Deep learning

Deep learning is a subset of machine learning methods focusing on learning representations of data, often expressed in terms of other, simpler representations, which aim is to solve a given target task [29]. Its approach to the problem of representation learning has proven to be effective in solving problems perceived as difficult in the fields of computer vision and natural language processing, among others.

Most modern deep learning solutions are based upon *deep neural networks*.

3.3.1 Deep neural networks

Deep neural networks are machine learning models which utilise multiple hidden layers in their internal infrastructure in order to produce a highly accurate representations of data.

3.3.2 Recurrent neural networks

Recurrent Neural Networks (or *RNNs*) are neural networks which work on arbitrary-length sequences as opposed to fixed-size inputs. The input sequences are processed by iterating through their elements and maintaining a *hidden state* containing information relative to what's been already processed [12].

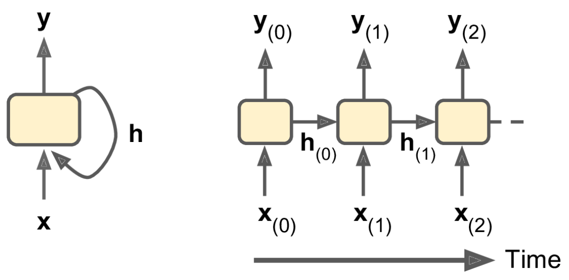

Figure 3.11: Memory cell's hidden state $h_{(t-1)}$ is passed as an input to the cell at the time step t, allowing the cell to take account its previous inputs when producing an output $y_{(t)}$. A recurrent cell can be *unrolled*, which means, that it is represented against a time axis, as shown in this example [26].

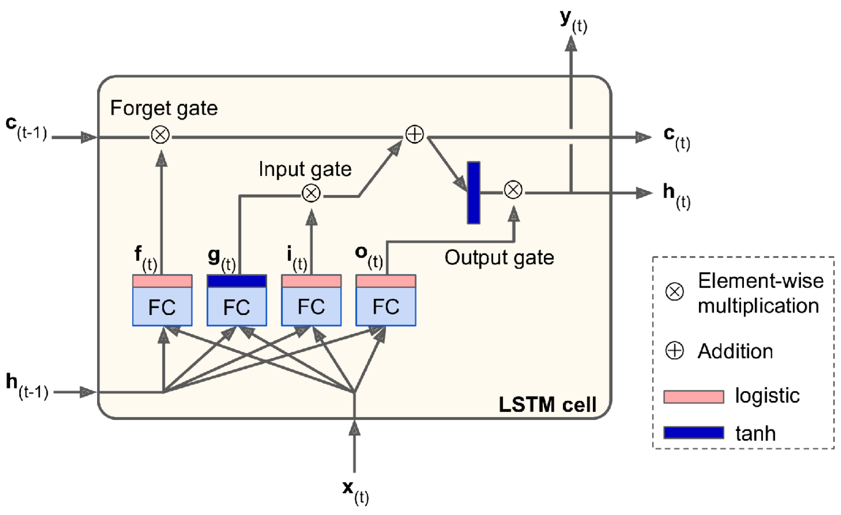

Figure 3.12: A LSTM cell through an introduction of both a short-term and long-term state, allowing the neural network to take into account from different time steps t_i depending on their determined importance [26].

RNNs are used in natural language processing tasks such as text translation or sentiment analysis, *time series* analysis and content generation [31]. Unlike feedforward neural networks, RNNs have connections pointing backwards.

Memory cells

Memory cells are basic building blocks of recurrent neural networks. At an each time step t a single memory cell outputs its hidden state $h_{(t)}$, which is a function of an input at that time step and the cell's state at the previous time step

$$h_{(t)} = f(h_{(t-1)}, x_{(t)}). \tag{3.20}$$

The cell's output at time step t, denoted as $y_{(t)}$ is also a function of the previous state and current inputs.

Long Short-Term Memory Cell

Long Short Term Memory Cell (or *LSTM*) is a memory cell whose internal state is split into two vectors: $h_{(t)}$, representing a short-term state, and $c_{(t)}$, which represents a long-term state [39].

A RNN learns whether an information should be kept, removed or read from $c_{(t)}$ in a LSTM cell. At each time step t an input long term state from a previous time step, $c_{(t-1)}$:

1. passes through a *forget gate*, which controls what information should be removed from $c_{(t)}$,
2. has an addition operation performed on it based on the output of an *input gate*,
3. is returned as a long term state $c_{(t)}$ of the cell at a current time step.

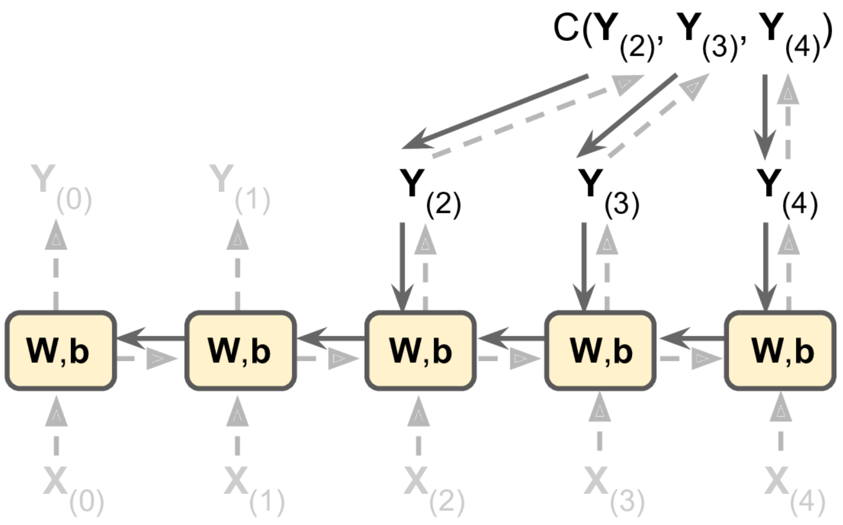

Figure 3.13: An illustration of the backpropagation through time strategy applied to an unfolded RNN. Note that the gradient is computed for all outputs used by the cost function (in this instance Y_2, Y_3 and Y_4), not just the final output of a network [26].

After the addition operation $c_{(t)}$'s state is copied and passed through an tanh function, which result is then modified by an *output gate*. These operations produce a short-term state $h_{(t)}$ equal to the cell's output for a current time step, denoted as $y_{(t)}$.

Also, at each time step t an input vector $x_{(t)}$ is passed along with $h_{(t-1)}$ through four fully-connected layers:

1. A layer outputting $g_{(t)}$, which is computed based on $x_{(t)}$ and $h_{(t-1)}$'s values,
2. A forget gate controller layer ($f_{(t)}$) controlling which parts of the long-term state to erase,
3. An input gate controller layer ($i_{(t)}$), which determines what parts of $g_{(t)}$'s output are added to $c_{(t)}$,
4. An output gate controller layer ($o_{(t)}$), determining which parts of $c_{(t)}$ should be output at t.

The LSTM cell can be thus defined mathematically as:

$$\begin{aligned} i_t &= \sigma(W^i x_t + U^i h_{t-1}) \\ f_t &= \sigma(W^f x_t + U^f h_{t-1}) \\ o_t &= \sigma(W^o x_t + U^o h_{t-1}) \\ g_t &= tanh(W^c x_t + U^c h_{t-1}) \\ c_t &= i_t \odot g_t + f_t \odot c_{t-1} \\ h_t &= o_t \odot tanh(c_t), \end{aligned} \tag{3.21}$$

where $W^i, W^f, W^o, U^i, U^f, U^o, U^c$ are weight matrices and $\odot$ is an element-wise multiplication.

Backpropagation through time

A recurrent neural network is trained using a strategy called the *backpropagation through time* (or *BPTT*), which is a regular backpropagation algorithm applied to a RNN unfolded through time.

Similarly to previously discussed algorithm, the forward pass is performed through the unfolded RNN, represented on a figure 3.13 by the dashed arrows. Then, an output sequence is evaluated using a cost function:

$$C(Y_{(t_{min})}, Y_{(t_{min}+1)}, \ldots, Y_{(t_{max})}), \tag{3.22}$$

where t_{min} and t_{max} represent first and final time steps, excluding ignored outputs, and the gradients of C are propagated backward through the unrolled network (step marked in figure 3.13 as solid arrows). The computed gradients are used to update the model's parameters θ.

Chapter 4

Recurrent neural networks-based natural language processing and document sentiment classification

Recurrent neural networks' ability to process arbitrary length inputs and interpret information based on a given context makes them an auspicious base for modern natural language processing systems. In fact, most of the state-of-the-art NLP applications, including document sentiment classification are based on RNNs [26].

However, this has not always been the case. Until 2013, recurrent neural networks had been thought to be difficult to train [78, 90] and most of the latter approaches required large datasets and prolonged computational time in order to converge [40].

In this chapter we discuss some of the key milestones in the field of neural network-based natural language processing, which have resulted in addressing aforementioned shortcomings and proliferation of cutting-edge NLP solutions. We also explain core concepts behind modern NLP frameworks and tool-kits, such as *tokenization*.

Finally, this chapter provides a perspective on the significance of introduced developments in the context of a task of document-level sentiment classification and the entire field of natural language processing. In this process, some of the established methods are introduced, including ULMFiT.

4.1 Document representation

In order for a text document to be processed by a neural network it has to be converted into a numerical representation. Most often such a document is represented by a vector obtained through a process called *text vectorization* [12].

4.1.1 Tokenization

Before a given text document can be vectorized, it has to be first segmented into individual subsequences, such as individual words, characters, or groups of multiple consecutive words or characters known as *n-grams* [12]. Such a task is referred to as *tokenization*.

Individual subsequences produced by a *tokenizer*, a system performing the tokenization, are called *tokens*. An example illustration of a tokenization process for a sentence in English is shown in a figure 4.1. A set of all the tokens generated by the tokenizer is called a *token vocabulary*, denoted as V^*.

More advanced tokenization methods, such as *Byte-Pair-Encoding segmentation* have been shown to increase a performance on many tasks, such as machine translation [53].

In highly inflected languages[1] (see section 2.5) a properly chosen tokenization method helps to reduce the

[1] Latin, Polish or Finnish are examples of highly inflected languages.

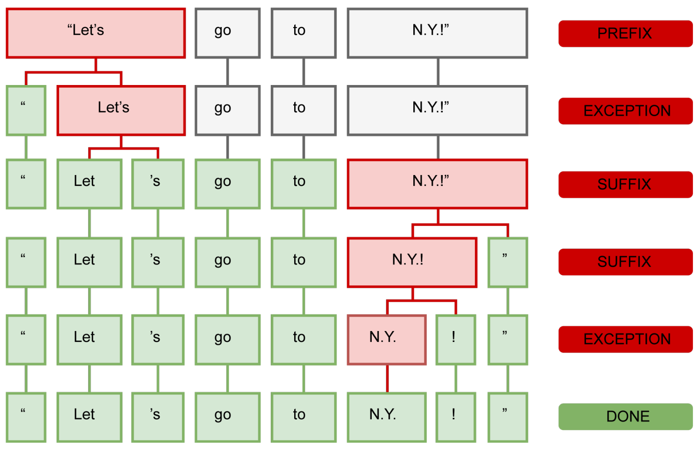

Figure 4.1: An illustration of the tokenization process performed by a *spaCy* tokenizer [77]. A raw text (*"Let's go to N.Y.!"*) is first split into subsequences on whitespace characters and for each subsequence a check is performed against a tokenizer's predefined set of rules and exceptions. Here, *"Let's* is first split into *"* and *Let's* based on a prefix splitting rule. Then *Let's* is again split into tokens *Let* and *'s* according to a defined exception (in English, *Let's* is short for *Let us*). Similarly, *N.Y.!"* produces tokens *N.Y.*, *!* and *"*.

amount of tokens generated, increasing the mode's ability to interpret their meaning without increasing the size of the vocabulary, having a positive impact on model's complexity, training duration and efficiency.

4.1.2 One-hot encoding

Tokens represent a discrete, categorical features, which can be represented as a vector through a method referred to as *one-hot encoding* [27].

Using one-hot encoding, each token t_i in the vocabulary of size n is assigned an unique index $i \in \{0, 1, \ldots, n-1\}$. Then, for each t_i, a vector v of size n is created, where all the elements are 0, except for the i-th entry, which has a value of 1.

Vectors obtained through one-hot encoding are binary, sparse and highly-dimensional, making them computationally inefficient for larger vocabularies. Such an encoding also does not maintain any semantic relationship between input tokens and output vectors, resulting in a loss of valuable information which could be taken advantage of during the process of training a neural network [49]. Thus, another vectorization method called *word embeddings* is more commonly used.

4.1.3 Token embeddings

Token embeddings are an efficient, dense representation of tokens in which similar tokens have a similar encoding. An embedding is a dense vector of floating-point values, which are the weights learned during the process of training a neural network.

The dimensionality of token embeddings varies depending on the size of the dataset. The more dimensional the embeddings are, the more likely they are to depict relationships between used tokens, at a cost of requiring more training data in order to learn [15].

4.1.4 Token embeddings layer

A *token embeddings layer* is an input layer of a neural network that performs mapping from integer indices representing individual tokens to associated vector embeddings [15, 40].

Depending on an implementation, an embeddings layer may use as an input an integer index $i \in \{0, 1, \ldots, n-1\}$ assigned for each token t in a vocabulary V^* of size n [15] or a one-hot-encoded vector representation of t [84].

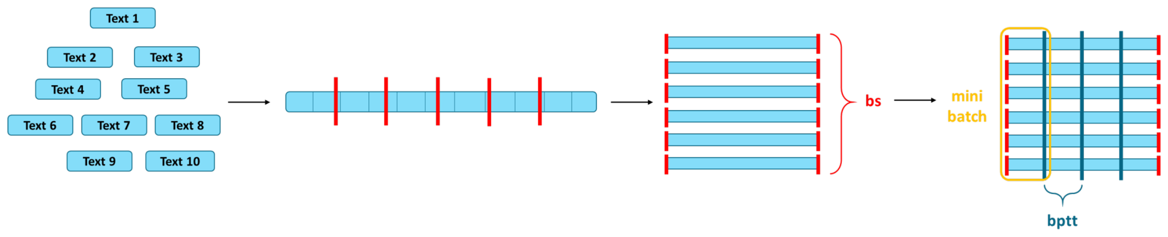

Figure 4.2: In a mini-batch-based process of training a RNN, a concatenated entity e of text documents is first split into a number of batches based on a defined batch size bs. As a result, batches consisting of n_t tokens are created, illustrated as a $bs \times n_t$ matrix. Then, mini-batches are created by truncating created batches based on a value of *bptt* parameter, which indicates, how many tokens are going to be back-propagated through when training the model. For instance, having a mini-batch consisting of 64 sequences of 70 tokens, during the training one mini-batch is fed to the model, and, as a result, all 64 sequences are processed in parallel. For each mini-batch model performs its predictions and calculates the loss before a backpropagation process begins [84].

4.1.5 Notation and assumptions on words and tokens

We will denote an individual word as w and a word vocabulary as V. Similarly, an individual token will be denoted as t, and the vocabulary of tokens as V^*. Words always convey a particular meaning, while the tokens are an output of a tokenizer [27]. The goal of tokens is to simplify a word and syntax-based meaning representation, through a generation of token-based representation.

For the purposes of simplicity, we will use the term words in the latter parts of this paper almost exclusively, unless it is important to make a distinction between them.

4.1.6 Corpora transformations

In natural language processing, a set of text documents used in a given learning task is referred to as *corpus*. Corpus can be either labelled or unlabelled, depending on a machine learning task it's applied to.

In order to use a text corpus for the purposes of training a recurrent neural network, it's contents, after tokenization, needs to be concatenated into a single entity. Such a process requires an addition of two tokens: a token signifying the beginning of a concatenated document, T_{BEG}, and a token put at the end of each concatenated document, T_{END}.

Defining C as a corpus and d_i as an individual member of the corpus, the resulting entity e is created:

$$e = (T_{\text{BEG}} \oplus d_1 \oplus T_{\text{END}}) \oplus (T_{\text{BEG}} \oplus d_2 \oplus T_{\text{END}}) \oplus \ldots (T_{\text{BEG}} \oplus d_n \oplus T_{\text{END}}), \tag{4.1}$$

where n is the length of the corpus (total number of text documents in a set) and $\oplus$ is the concatenation.

However, due to computational and memory constraints, training a RNN using a large corpora is performed using a mini-batch-based backpropagation through time algorithm (see figure 4.2) [84].

4.2 Language modelling

Language modelling is the task of learning a probability distribution over words in a given vocabulary V. As a result, a language model is created, which assigns probability for the likelihood of a given word (or sequence of words) to follow an arbitrary sequence of words [27].

Most often the aim of the language models is to compute the probability of the occurrence of word w_t given $n-1$ previous words in a sequence, denoted as $P(w_t|w_{t-1}, w_{t-2}, \ldots, w_1)$, where [81]:

$$P(w_t|w_{t-1}, w_{t-2}, \ldots, w_1) = P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\ldots P(w_t|w_1w_2\ldots w_t - 1). \tag{4.2}$$

Due to the fact that the last term of equation 4.2 requires conditioning on $n-1$ words, which is a task almost equally difficult as modelling an entire sentence, language models rely on the *Markov assumption*, assuming that the next word in a sequence depends only on the last k words (*"future is independent of past given the present"*) [27]:

$$P(w_i|w_{i-1}w_{i-2}\dots w_1) \approx P(w_i|w_i w_{i-1}\dots w_{i-k}). \tag{4.3}$$

The probability of a whole sentence or a document can be as such approximated by the product of probabilities of each word given n previous words [81]:

$$P(w_1, w_2, \dots, w_t) = \prod_i P(w_i|w_{i-1}w_{i-2}\dots w_{i-n+1}). \tag{4.4}$$

4.3 Transfer learning: from fine-tuning of pre-trained word embeddings to fine-tuning of pre-trained language models

Most of the earlier research on transfer learning applications in neural network-based natural language processing focused on transductive transfer learning, where, for instance, the models' ability to perform accurate predictions in classification tasks when documents from a different domain were used as an input was measured [5, 40].

For inductive transfer learning, two noteworthy techniques have been proposed: first, a fine-tuning of pre-trained word embeddings method was proposed [63], and later - fine-tuning of entire, pre-trained models [40].

4.3.1 Fine-tuning pre-trained word embeddings

In order to improve the word embeddings' ability to represent semantic relationships between words in a given vocabulary V, highly-specialised machine learning models, such as *word2vec* and *GloVe* have been used, which, due to a substantial improvement of their performance, have allowed to be trained on much larger datasets, thus obtaining relation-maintaining word embeddings [56, 63, 72].

Applied in an embedding layer of a neural network, pre-trained word embeddings have been shown to increase the model's performance on various NLP tasks [111, 54].

However, as the usage of pre-trained word embeddings affects only the first layer of a model, the remaining parameters still remain randomly initialised. Taking into account the benefits of pre-training an entire model, which include an improved generalisation [22], first successful language model pre-training and fine-tuning methods have been proposed. Their impact on neural networks-based natural language processing cannot be understated [82].

4.3.2 Fine-tuning pre-trained language models

Language models had been historically thought to be prone to suffering from a phenomena known as *catastrophic forgetting* [28] when fine-tuned with a classifier, as well as requiring large datasets in order not to overfit [17]. Nowadays, they are ubiquitous in natural language processing, allowing for state-of-the-art results to be achieved on a wide range of tasks [82].

This is due to the fact that novel methods, such as the ULMFiT [40], *BERT* [20] or *GPT2* [75] have made it possible to perform a successful pre-training and latter fine-tuning of a deep neural networks-based language model.

Language modelling as a pre-training source task has also been shown to lead to a better performance on target task, as opposed to machine translation or auto-encoding [113], especially when syntactic information is of relevance.

Arguably, the most important benefit of using a pre-trained language model is that it substantially reduces the amount of labelled training examples needed for a target model to converge, as shown in figure 4.3.

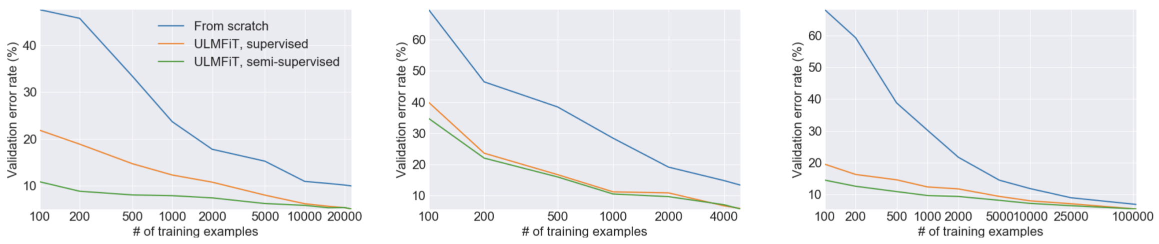

Figure 4.3: A comparison of validation error rates for supervised and semi-supervised ULMFiT method applications as opposed to training an AWD-LSTM model from scratch, with varying amount of training examples. Tests were performed on *IMDB* (left), *TREC-6* (center) and *AG* (right) datasets [40].

It is, however, computationally expensive to pre-train a language model which reaches a low *perplexity*[2] which is a measurement of how well a probability distribution or probability model predicts a sample [82].

4.4 AWD-LSTM

Due to over-parametrisation of neural networks, their generalisation ability depends heavily on the regularisation techniques used. However, naive applications of strategies, such as dropout and batch normalisation in recurrent neural networks have not been highly successful, compared to results in feed-forward and convolutional neural networks.

A proposal of *AvSDG Weight-Dropped LSTM* (or *AWD-LSTM*), a set of regularisation strategies developed for RNNs [61] has helped to produce state-of-the-art results in language modelling [84].

Model	Validation	Test
AWD-LSTM	68.6	65.8
- variable sequence lengths	69.3	66.2
- embedding dropout	71.1	68.1
- weight decay	71.9	68.7
- AR/TAR	73.2	70.1
- weight-dropping	78.4	74.9

Table 4.1: An impact of removal of regularisation techniques used in AWD-LSTM on the model's validation and test perplexity measured on a *WikiText-2* dataset. A removal of hidden-to-hidden weights regularisation performed by weight dropping alone leads to an increase of over 10 points of perplexity as compared to a model employing all the regularisation strategies, proving that regularisation has a significant, positive impact on the recurrent neural network's performance [61].

4.4.1 Weight-dropped LSTM

AWD-LSTM makes use of *DropConnect* on the recurrent, hidden-to-hidden weight matrices (U^i, U^f, U^o, U^c), as opposed to previous techniques, which introduced dropout between time steps, acting on hidden state vector h_{t-1}, or on update of a memory state c_t.

This enables the use of external, highly-optimised LSTM implementations, such as *cuDNN LSTM* [61], while retaining the LSTM's ability to learn long term dependencies of input data [86].

4.4.2 Variable-length backpropagation through time

Having a fixed length of sequences in a mini-batches used to train the network using the backpropagation through time algorithm results in an inefficient use of available training data. Merity et al. illustrates

[2]Perplexity is a measurement of how well a probability distribution or probability model predicts the sample. The perplexity of language model can be understood as the level of perplexity when predicting the following word - a total number of choices a language model can make [41].

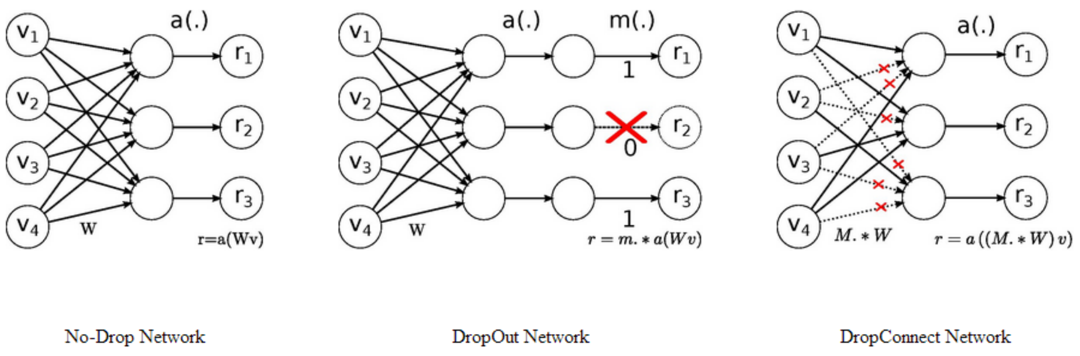

Figure 4.4: DropConnect sets a randomly selected subset of weight values in a neural network to zero, as opposed to a regular dropout, which is applied as a mask, denoted here as $m(\cdot)$, on a random subset of network's activations $a(\cdot)$ [94].

this by an example: having 100 elements to perform a backpropagation with a fixed *bptt* window of 10, any element divisible by 10 will not have any element to backpropagate into [61], essentially meaning, that for an every final element of a mini-batch a language model's prediction is never performed. To prevent such an efficiency, an use of *variable-length backpropagation sequences* was proposed.

Using variable-length backpropagation sequences approach, a base sequence length is determined to be $bptt$ with probability p and $\frac{bptt}{2}$ with probability $1 - p$. Then, a sequence length l is selected according to $\mathcal{N}(bptt, s)$, where s is the standard deviation and $\mathcal{N}$ - a normal distribution [86]. This results in a more efficient usage of available training data.

4.4.3 Variational dropout

Variational dropout addresses the inefficiency of applying a regular dropout in long short-term memory networks, by performing a generation of a dropout mask once and then re-using a generated mask within a single forward and backward pass, as opposed to its re-generation on every dropout function call . This ensures that in an RNN at each time step t an identical dropout mask is applied for input x_t.

In AWD-LSTM, variational dropout is used in all dropout operations, excluding those related to hidden-to-hidden weight matrices described in section 4.4.1. Specifically, a variational dropout mask is applied for all inputs and outputs of LSTM [61].

4.4.4 Embeddings dropout

Another regularisation called *embeddings dropout* is used in AWD-LSTM to perform a dropout on a randomly-chosen set of inputs to the embedding matrix of a neural network, resulting in some of the input tokens being omitted during a single forward and backward pass process of training a network.

Intuitively, the goal of using an embeddings dropout is to make a model less prone produce output based on the presence of individual tokens. Used with variational dropout, embeddings dropout method has been shown to improve the performance of LSTM-based natural language processing models performing language modelling [61] and sentiment analysis [24], among others.

4.4.5 Activation regularisation and temporal activation regularisation

An *activation regularisation* is a L2 regularisation applied on the individual activation values of the neural network in order to penalise activations significantly larger than 0 as a mean of regularising the network. It is defined as:

$$AR = \alpha \cdot L2(m \odot h_t), \tag{4.5}$$

where m is the dropout mask, h_t is the output of the recurrent neural network at timestep t and α is a scaling coefficient.

To prevent substantial changes between hidden state values at subsequent time steps, a *temporal activation regularisation* is used:

$$TAR = \beta \cdot L2(h_t - h_{t+1}), \tag{4.6}$$

where β is a scaling coefficient. Both TAR and AR are applied only to the output of the final RNN's layer [62, 61].

4.5 Adam optimiser

Gradient descent-based optimisation methods are by far the most common way to optimise neural networks. However, an usage of plain mini-batch gradient descent algorithm (introduced in section 3.1.6) is not ideal due to the fact that it is often difficult to choose a proper learning rate: small η leads to a slow convergence, while high η might cause the loss function to fluctuate around its local/global minimum, or even to diverge.

An *Adaptive Moment Estimation* (or *Adam*) is a SDG-based method that computes adaptive learning rates for each parameter [48]. In Adam, an exponentially decaying average of past gradients, denoted as m_t:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{4.7}$$

which estimates the first moment of the gradient (the mean) is stored, along with an exponentially decaying average of past squared gradients, defined as v_t:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \tag{4.8}$$

estimating the second moment (the uncentered variance) of the gradient [80].

It has been observed that both m_t and v_t initialised as vectors of zeros are biased towards zero, especially during the initial time steps, and when the decay rates are small (β_1 and β_2 are close to zero). To counteract this bias, bias-corrected first and second moment estimates are computed:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t}, \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t}, \end{aligned} \tag{4.9}$$

in order to be used to update the parameters of the model [80]:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t. \tag{4.10}$$

An usage of default parameters of $\beta_1 = 0.99$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ has been proposed by authors of the method [48].

4.6 Impact on natural language processing

While different methods of pre-training word embeddings have been extensively used in many deep learning-based NLP models with an effect of improved performance by 2-3 percentage points [47], they have been able to affect only embedding layers of neural networks, leaving the majority of network parameters randomly initiated [40].

This requires a model using word embeddings not only to learn from scratch how to disambiguate words, but also how to derive meaning from a sequence of words, modelling complex phenomena such as *com-*

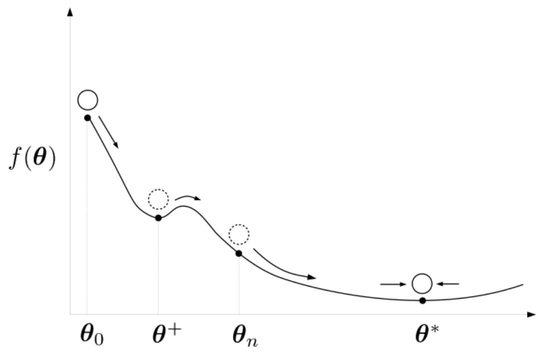

Figure 4.5: Adam is an example of a *Heavy Ball with Friction* (*HBF*) method. It's averaging over past gradients corresponds to a velocity that results in the optimiser being "resistant" to be pushed into small regions. Adam has been shown to overshoot small, local minima of functions and be able to find flat minima which result in a improved generalisation of an optimised model, as shown in this illustration [37].

positionality[3], *polysemy*[4], *anaphora*[5], long-term depedencies, agreement, negation, and many more core aspects of natural language understanding [44]. As a result, the model's performance still depends on a large set of training examples available, limiting usefulness of the method when task-specific data sources are scarce.

An emergence of *ELMo* [73], ULMFiT [40] or *OpenAI transformer* [74] among others has radically changed the NLP landscape, successfully demonstrating that pre-trained language models can be used to achieve state-of-the-art results in the field. The impact of a paradigm shift from using pre-trained word embeddings to pre-trained language models has been as significant to the field of natural language processing as the introduction of models pre-trained on ImageNet [18] for computer vision [44], resulting in a substantial improvement of target models' performance [36, 82]. In case of ULMFiT, over ten times less labelled examples are needed for the target classifier model to achieve similar performance to a non-pre-trained model (see Figure 4.3) [40].

It would not be possible for language models to make such an impact on the machine learning-based NLP field if it weren't for improvements in the area of tokenization (section 4.1), neural network regularisation, and optimisation, among others. For instance, ULMFiT relies heavily on the work of Merity et al. on AWD-LSTM (section 4.4), implementing its regularisation strategies, as well as Adam stochastic optimisation (section 4.5).

Latest pre-training methods, such as *RoBERTa* [59] or *MultiFiT* [21] prove that the performance of language modelling can be still performed through an experimentation with different neural network infrastructures, amount of unsupervised training data, cross-lingual language modelling, and more [82].

4.7 Impact on document-level sentiment classification

When a neural networks-based model is used in a natural language processing task, including the domain-level sentiment classification, it is required for a model to obtain knowledge about the language, ranging from low-level (such as meaning of words) to high-level (e.g. that the dog barks) [76].

Such a knowledge can be obtained by training a neural network on a large set of domain-specific documents for a given target task. Unfortunately, obtaining high accuracy this way has often involved datasets consisting of tens of thousands, if not hundreds of thousands of in-domain documents. This has proven to limit neural networks-based NLP applications to tasks with relatively large amount of labelled training data available [40]. For a problem of document-level sentiment classification, such a requirement has

[3]The principle of compositionality is the principle that the meaning of a complex expression is determined by the meanings of its constituent expressions and the rules used to combine them [107].

[4]Polysemy is the capacity for a word or phrase to have multiple meanings, usually related by contiguity of meaning within a semantic field [106].

[5]Anaphora is the use of an expression whose interpretation depends upon another expression in context [98].

limited neural networks-based implementations to languages, such as English, for which a vast amount of qualitative opinion sources, including *Amazon* or IMDB reviews could be found.

An advent of methods for inductive transfer learning-based pre-training and fine-tuning of language models has, however, fundamentally changed the NLP landscape, leveraging widely available, unlabelled sets of textual information in virtually every language, such as Wikipedia, as means of learning both low-level and high-level representations of data (see section 4.6).

ULMFiT, being a notable example such a technique, utilises LSTM recurrent neural networks-based language models (section 3.3.2) with applied AWD-LSTM regularisation techniques, such as variational dropout, weight-dropped LSTM or embeddings dropout (section 4.4), used in an inductive transfer learning process (section 3.1.4) of domain-general dataset-based language model pre-training, and latter, domain-specific dataset-based language model fine-tuning (see section 4.3.2) and final training to produce a text classifier, which achieves performance comparable to training an entire RNN model from scratch while requiring up to twenty times less labelled training data [82, 40].

As such, a language model-based document sentiment classification system pre-trained on general, unlabelled data (such as encyclopedic documents), fine-tuned on domain-specific unlabelled opinions (often easier to find in greater amounts than labelled ones), and finally trained on just thousands of target labelled opinion examples, can achieve a high overall performance of over 95 % on less than 10000 labelled examples [76, 55, 59, 58]. This means that by leveraging language model pre-training and fine-tuning, neural networks-based document sentiment classification can be performed on a variety of new languages and domains, previously limited by scarcity of available labelled data.

An additional benefit of using pre-trained language models is that, once trained, they can be re-used for a wide variety of NLP tasks from different domains. Having pre-trained a Polish domain-general language model on a set of Wikipedia documents, we can use it for purpose of numerous NLP tasks, including document sentiment analysis and any text classification. When a goal is to prepare separate document sentiment classifiers for a variety of Polish product review sources (e.g. Filmweb, Allegro, Empik), a previously prepared domain-general language model can be fine-tuned on data available from all these sources, and then multiple classifiers for each respectable source can be produced using available labelled data respectively. In addition to that benefit, having produced fine-tuned language model, it takes only several minutes for a resulting, highly accurate classifier to be produced, as shown in chapter 7 of this work.

In conclusion, document-level sentiment classification task can benefit greatly from the use of effective methods of pre-training and fine-tuning language models introduced in this chapter for a multitude of reasons:

- They allow for a ten- to twenty-fold decrease of amount of labelled training data needed to achieve performance comparable with models trained from scratch by leveraging a plethora of unlabelled text document data sources available for a variety of domains and languages,
- They allow for an reuse of domain-general and domain-specific language models for multiple tasks, including, but not limited to tasks in the field of sentiment analysis,
- They allow for target document sentiment classifiers to be produced in only several minutes of training time.

As such, a design and an implementation of a document-level sentiment classification system discussed in depth in chapter 5 of this work leverages aforementioned techniques as means of improving the performance on a target classification task.

Chapter 5

Design and implementation

In this chapter we introduce a design and am implementation of a recurrent neural network-based, document-level text sentiment classification system.

The classifier is produced using an Universal Language Model Fine-Tuning method [40]. For that purpose, we pre-train a *domain-general Polish language model* using a large set of general domain data, which, when fine-tuned with unlabelled target domain data of choice, results in a *domain-specific language model.* Finally, the resulting *target classifier* is created by an addition of two linear encoding layers and training on labelled target domain data. In this process a `fast.ai`[1] deep learning library is used.

For the purpose of this work, a target domain of movie reviews has been chosen. Target domain data are obtained from a prepared *Filmweb+* dataset, and general domain data - from a set of articles from Polish Wikipedia, *plwiki.*

To increase the accuracy of a resulting classifier, some of the novel methods in the field of neural networks-based natural language processing will be used, including the `sentencepiece` tokenizer.

5.1 General system requirements

The resulting sentiment classification system is required, given a path to an input text document containing an opinionated statement in Polish, to output either a `positive`, `neutral` or `negative` label based on the contents of the document, along with a computed probability of input belonging to the `positive` and `negative` class.

The accuracy of the resulting system in the classification task should be state-of-the-art comparable [16, 50].

For convenience, the system should allow for multiple of documents to be processed at once.

5.2 System design requirements

It was determined that in order to meet the aforementioned requirements, the resulting sentiment classification system should meet the functional and non-functional requirements defined below.

5.2.1 Functional requirements

The sentiment classification system:

1. Should process either a single text document or multiple text documents stored in `.txt` given a path to a single text document or a path to a directory containing multiple text documents,
2. Should allow for an user interaction using a command-line interface (`CLI`),
3. Should be based upon a recurrent neural networks-based machine learning model, which allows for arbitrary-length sequences to be processed,

[1]https://www.fast.ai/

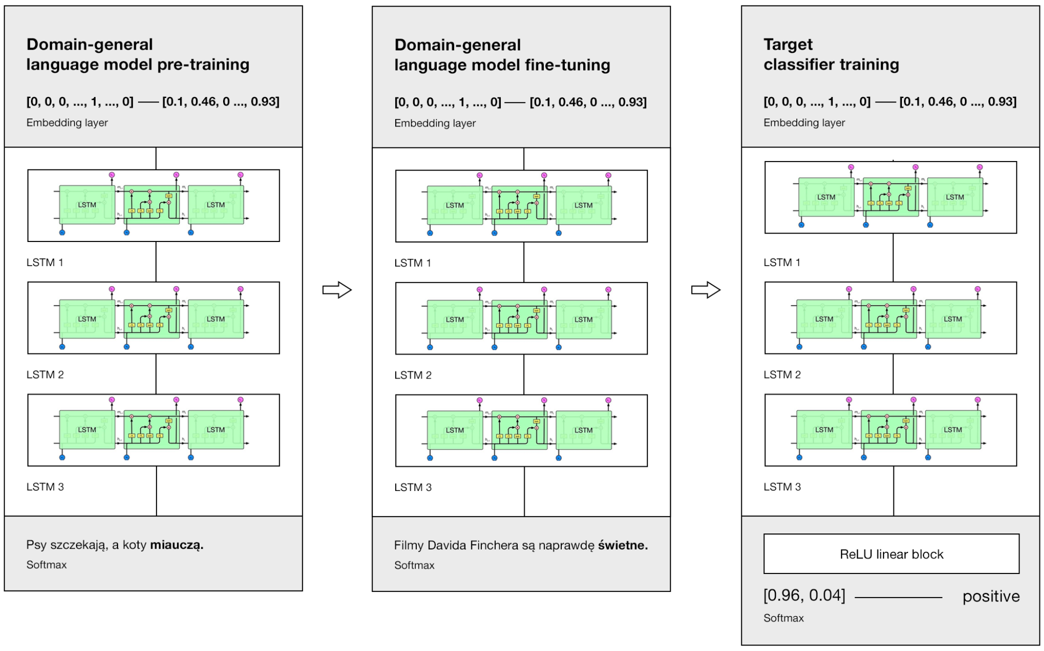

Figure 5.1: In ULMFiT, a domain-general language model is fine-tuned, resulting in a domain-specific language model, which then, augmented with two additional linear blocks and trained on labelled training data produces a target, domain-specific classifier.

4. Should be based on fine-tuned, pre-trained language models, allowing for a smaller amount of labelled data to be used for training a target classifier,
5. Should leverage available machine learning frameworks and libraries,
6. Should be trained in less than 24 hours of computing time using commercially-available consumer hardware,
7. Should allow for the underlying neural network model to be trained from scratch using provided input parameters for the purposes of evaluating prepared models.

5.2.2 Non-functional requirements

1. The system should run as an application on a latest version of `macOS` operating system or a compatible `Linux` distribution with `Nvidia CUDA` drivers available,
2. The system should be capable of working on a computer system equipped with a `CUDA`-compatible graphics card with at least `11 GB` of onboard memory.

5.3 System design

The resulting system has been designed to consist of several parts:

1. A `models` library, containing methods performing a pre-training and fine-tuning of language models, as well as training final classifiers using provided hyperparameters and input datasets,
2. A `preprocessors` pre-processing library, providing methods used for pre-processing of used datasets,
3. A `databunches` library, used to generate `DataBunches` from pre-processed data,
4. A `fincher` training and predictions library, handling classification of a given input document or multiple given input documents based on provided path, as well as managing an entire data pre-processing and model training flow, using methods available in `models, preprocessors` and `databunches` libraries,

5. A `command-line interface`, parsing arguments provided by the user and leveraging available library methods for the purposes of:
   (a) Performing a sentiment classification for a given input text document (or multiple input text documents),
   (b) Generating databunches for the purpose of different training set-ups,
   (c) Training a target classifier used for latter predictions based on provided input data.

A detailed documentation of methods and classes used in the document sentiment classification system is available in a repository containing the system source code (see appendix A).

5.3.1 Training library

The Universal Language Model Fine-Tuning method first proposed by Howard et al. [40] has been chosen as a basis for the `training library`. This choice is motivated by meeting all of the aforementioned functional system requirements, as well being supported by a documented [13], `PyTorch`-based `fast.ai` library.

5.3.2 Command-line interface

A command-line interface executes methods defined in the `training library` having processed input data provided by the user. It is operated according to a provided user manual.

5.4 Motivation of ULMFiT choice

The ULMFiT method has been proven to produce accurate classifiers when used on tasks in different foreign languages [92], including Polish, as well as long, real-world text documents [40].Since a majority of work on classification of text documents in Polish with ULMFiT has been done on datasets consisting of short text documents only, such as user posts fetched from *Twitter* [16, 50], an exploration of such an ability for Polish texts might be of value for a future reference.

5.5 Datasets

In order to obtain a domain-general Polish language model, a plwiki dataset is used. Target domain-specific classifier is trained on the Filmweb+ dataset.

5.5.1 Filmweb+

Filmweb+ dataset contains 26890 Polish movie reviews fetched from Filmweb, Film.org.pl, Kinoblog, FDB and BlogFilmowy24 websites. Among these reviews, 7655 are unlabelled and 19235 are labelled as either positive, neutral or negative. There are 708629 unique words in the dataset, with an average of 514 words (or 3582 characters) per document.

Filmweb+ dataset	
Class	# of reviews
positive	6860
neutral	9332
negative	3043
unlabelled	7655

Table 5.1: A count of documents in a Filmweb+ dataset. Labelled text documents' count is divided by positive, neutral and negative classes available.

Labelled reviews are assigned to a class based on the score from range of $[1, 10]$, given by a reviewer to a movie alongside a posted review, based on a criteria presented in Table 5.2.

Movie score	Label
$[1, 4]$	negative
$[5, 7]$	neutral
$[8, 10]$	positive

Table 5.2: Criteria used for assigning a document to a category based on a reviewer's movie score in a Filmweb+ dataset.

In practice, neutral reviews (scored in a range of $[5, 7]$) often belong to either the positive or negative category based on their sentiment towards the movie. Thus, to produce a labelled dataset of reviews used to train the final classifier, 3043 negative and 3043 randomly chosen positive reviews are used.

5.5.2 plwiki

Based on the contents of a Polish Wikipedia, plwiki is a dataset consisting of 110038 unlabelled articles with a minimum text length of 2800 words, excluding templates and disambiguation pages. It contains an average of 1088 words per article and 1887767 unique words in the dataset. In total, plwiki contains 119754538 words.

Articles in plwiki contain encyclopedic information on a wide range of topics, including history, geography, science, sports, biology and philosophy.

5.6 Preparations

Contents of the prepared datasets need to be tokenized, numericalised and processed into individual entities (see section 5.5) in order to be used during the process of training. For this purpose, we use a `sentepiece`-based tokenization and `Databunches` available in a `fast.ai` library.

5.6.1 Sentencepiece-based tokenization

Being a highly-inflected language, Polish requires an use of efficient tokenization method in order to reflect the word structure found in our corpora.

For this reason, an unsupervised `sentencepiece`[2] tokenizer is used to convert a word-based document representation into a token-based one, which produces subword-based tokens using an unigram segmentation algorithm [53], found to be more efficient in similiar tasks than a character-level and word-level tokens [16].

The output of a sentencepiece-based tokenizer varies depending on the size of the vocabulary V^* chosen, indicating a total number of unique tokens to be used to represent our input text documents.

To indicate certain features of input text documents special tokens are also added in a tokenization process [13]:

1. `UNK` (xxunk), for an unknown word (one not present in the token vocabulary V^*),
2. `PAD` (xxpad), used for padding, if texts of different length are found in a single batch,
3. `BOS` (xxbos), which represents the beginning of a new document in a dataset,
4. `EOS` (xxeos), which indicates the end of a document in a dataset,
5. `TK_MAJ` (xxmaj), indicating that the next word begins with a capital in the original document,
6. `TK_UP` (xxup), indicating that the next word is written in all caps in the original document,
7. `TK_REP` (xxrep), indicating that the next character is repeated n times in the original document (`xxrep n char`),
8. `TK_WREP` (xxwrep), indicating that the next word is repeated n times in the original document (`xxwrep n word`).

[2]https://github.com/google/sentencepiece

5.6.2 Databunches

A `Databunch` object in `fast.ai` library handles the pre-processing of a dataset and splits it into training, test and validation sets divided into batches of a given `batch_size`, allowing for them to be used to train a neural network model.

In order to train a target sentiment classifier, three separate databunches are first created:

1. A general `TextLMDataBunch` for plwiki dataset,
2. A domain-specific `TextLMDataBunch` for unlabelled Filmweb+ dataset,
3. A target classifier `TextClasDataBunch` for labelled Filmweb+ dataset.

Data for all of these databunches is pre-processed by a sentencepiece-based tokenizer (see section 5.6.1), which generates a vocabulary of tokens on a domain-general dataset and then re-uses it on a domain-specific and target classifier datasets, and a numericaliser (see section 4.1), responsible for a conversion of tokens into a one-hot encoded vector, neural network embedding layer's input format.

A `TextLMDataBunch` also handles the process of producing labels used for training a language model, and a `TextClasDataBunch` uses available labelled data to train the classifier.

5.7 Domain-general language model pre-training

A domain-general language model is a long short-term memory recurrent neural network (see section 3.3.2) based on AWD-LSTM (see section 4.4) with an embedding of size 400, three hidden layers, 1150 hidden activations per layer and a BPTT batch size of 70 (see section 4.4.2) [40].

The model makes an extensive use of regularisation strategies proposed by Merity et al. [61]. A variational dropout (see section 4.4.3) of 0.3 is applied to recurrent layers, 0.4 to non-recurrent layers of the network and 0.05 to non-input embedding layers (resulting in zeroing of individual embeddings) . Also, an embeddings dropout of 0.4 is used in input embedding layers (see section 4.4.4) and an weight dropout of 0.5 is utilised on hidden-to-hidden weight matrices (see section 4.4.1).

In order to improve the generalisation performance and cut down on the amount of training time required for the model to converge, a *one-cycle* policy is used, being an improvement over the *slanted-triangular learning rates* policy proposed by Howard et. al [40].

5.7.1 One-cycle policy and superconvergence

In one-cycle policy, a learning rate is increased up to a defined point of `max_lr`, followed by a decay to a starting point of `min_lr` [33]. This process performed over a given number of training *epochs*[3].

It has been observed that high learning rates act as a mean of regularising a neural network, preventing overfitting and allowing a used optimiser to find flatter minima of the functions. Also, a linear decrease of momentum used by an optimiser from a higher value to a lower one during the increase of a learning rate, followed by an increase back to the starting point when the learning rate is decreased has been found to increase the performance of the network [33].

A usage of one-cycle policy allows for larger learning rate values during the process of training, significantly reducing the amount of computational time it takes for a model to converge while at the same time preventing the occurrence of overfitting. This phenomena is referred to as *superconvergence* [88].

5.7.2 Our language model pre-training set-up

Our domain-general language model is trained on a *plwiki* dataset using a one-cycle policy policy for 10 epochs with a `batch_size` of 64. The maximum learning rate is a constant numeric value found using a `lr_finder` [32] multiplied by a factor of `batch_size/48`. The momentum's value changes from a `mom_min` value of 0.7 to `mom_max` value of 0.8 during training, as shown in a rightmost chart in figure 5.2.

While pre-training a domain-general language model is the most computationally expensive part of an entire classifier preparations process, it allows to take an advantage of benefits discussed in section 4.3.2.

[3] An epoch is a full forward and backward pass performed on all the elements in a training set.

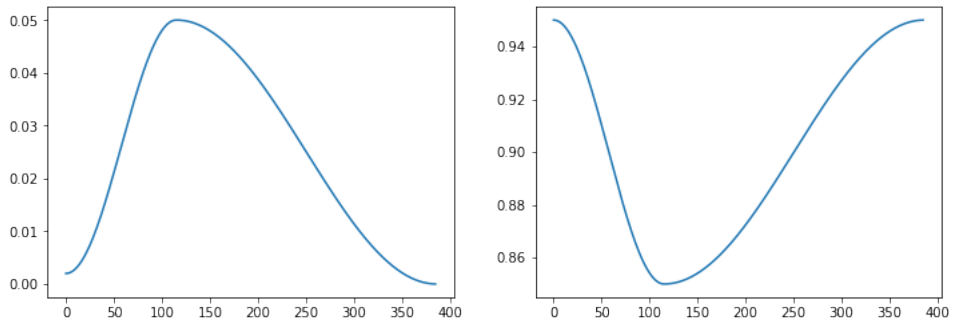

Figure 5.2: Based on unpublished work a variant of one-cycle policy is used where a learning rate is increased up to a `lr_max`, after which a cosine annealing is performed from `lr_max` to 0, as shown in the graph on the left. The momentum changes from `mom_min` to `mom_max` by following the symetric cosine seen on the graph on the right [13].

5.8 Domain-specific language model fine-tuning

To produce a domain-specific language model, a pre-trained, domain-general language model is fine-tuned on all available, unlabelled data from a target domain.

In order to reduce a risk overfitting, a *gradual unfreezing* together with a one-cycle policy is used when fine-tuning a language model.

5.8.1 Gradual unfreezing

Gradual unfreezing is a process where, contrary to fine-tuning all model layers at once during training, a model is fine-tuned by first preventing a change of weights (or *freezing*) of all the layers except the final one (containing the least general knowledge [112]) in a training iteration and then incrementally allowing for weight updates to preceding layers (or *unfreezing*) to occur [40].

5.8.2 Our language model fine-tuning set-up

Our domain-specific language model is obtained in a two-step process. Firstly, the final layer of our domain-general LM (see section 5.7.2) is fine-tuned using a two-epoch-long one-cycle policy with constant `max_lr` determined using `lr_finder`, again multiplied by a factor of `batch_size/48`. Secondly, all of the weights are unfrozen, and the entire model is fine-tuned again using a one-cycle policy, but with 8 total epochs. In the entire process, `mom_min` of 0.7 and `mom_max` of 0.8 are used, as well as a full Filmweb+ dataset, including both labelled and unlabelled examples, but discarding available labels, which are irrelevant for the task of language modelling.

5.9 Target classifier

In a final step of ULMFiT, a fine-tuned, domain-specific language model is used produce a target, domain-specific classifier. For this purpose, two additional *linear decoding* layers are added to the fine-tuned language model: an intermediate one with 50 ReLU activations, taking as an input *concat pooled* last hidden layer states, a and final, softmax-activated layer, outputting probability distribution over target classes. Both of these blocks use batch normalization [43] and dropout as means of regularisation [40].

A target classifier is trained using a gradual unfreezing, one-cycle policy and *discriminative fine-tuning*. An `TextClasDataBunch` provides labelled data during the process of training a classifier in mini-batches, where each mini-batch contains sequences consisting of an entire text document [84].

5.9.1 Concat pooling

Some of the input words in a document may contain information which is more relevant to the task of classification than others. However, after each time step t, when a single word from a document is processed, a hidden state h_t of the LSTM cell is updated, resulting in a potential loss of valuable

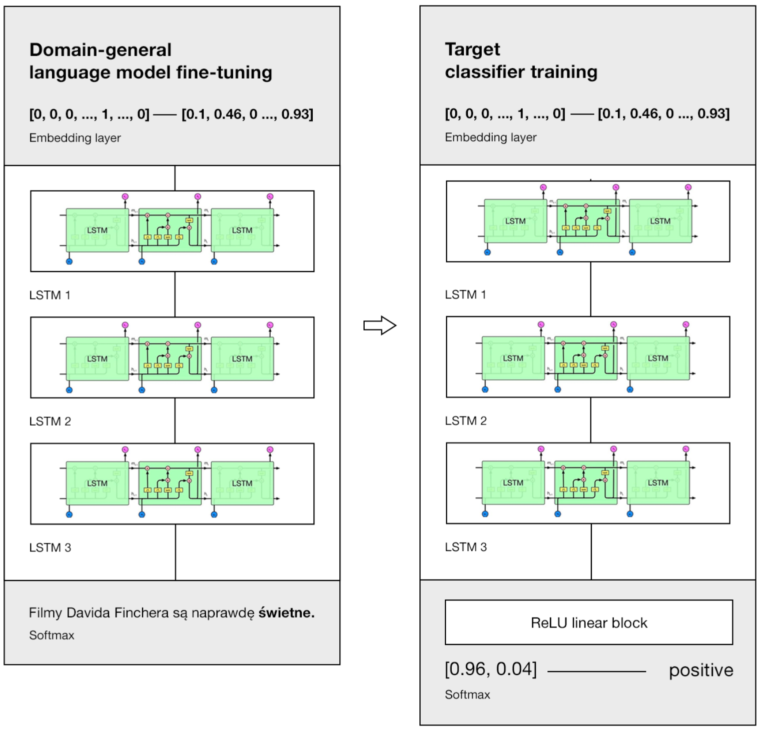

Figure 5.3: A target classifier is produced by an addition of two linear decoding layers to the existing, domain-specific language model.

information. For this reason, Howard et al. introduced a concat pooling technique, which performs a concatenation of last hidden layer's state h_t with a *max-pooled* and *mean-pooled* representation of the hidden state [40].

The max-pooled representation of h_t is generated from the largest values for each of the 400 embedding features over all hidden states $H = \{h_1, h_2, \ldots, h_n\}$ of the document in the final layer of the network, and the mean-pooled representation of h_t is prepared by computing an average of each embedding features [84].

As a result of concat pooling, a state h_c is returned:

$$h_c = [h_t, maxpool(H), meanpool(H)], \tag{5.1}$$

where [] is concatenation [40].

5.9.2 Discriminative fine-tuning

Using discriminative fine-tuning, a different learning rate parameter is used during training for each of the model's layer l, denoted as η^l. Howard et al. recommend to make a choice of η used to train a final layer first and then to apply $\eta^{l-1} = \eta^l/2.6$ for all preceding layers of the network based on their empirical study [40].

Group-discriminative fine-tuning

In practice, most of the ULMFiT implementations use a variant of discriminative fine-tuning, where a different learning rate parameter is used for a group of layers g, denoted as η^g [45] when trained using a one-cycle policy. We refer to this variant as *group-discriminative fine-tuning*.

While this behaviour is not explicitly documented in a `fast.ai` library, a group-discriminative fine-tuning is used whenever a Python `slice` object of format `(start, end, range)` is passed as `max_lr` argument to a `fit_one_cycle` method instead of a constant, numerical value.

Having `min_lr` as slice's `start` and`max_lr` as slice's `end`, a log-stepped array H of values from `min_lr` to `max_lr` in n steps is created, where each $\eta^{g_i} \in H$ is a learning rate to use for training a layer group g_i, and n is a total number layer groups in a neural network.

5.9.3 Our target classifier training set-up

We perform gradual unfreezing combined with group-discriminative fine-tuning and a one-cycle policy to train the target classifier. A starting learning rate lr of $2 \cdot 10^{-2}$ scaled by `batch_size/48` is chosen, and momentum varies from a `mom_min` value of 0.7 to `mom_max` value of 0.8. The entire training process, performed on a labelled subset of a Filmweb+ dataset, is shown in table 5.3. As a result, we obtain a final classifier model.

Target classifier training			
Step	Epochs	Frozen to layer	Maximum learning rate
1	2	-1	lr
2	2	-2	slice$(lr/(2.6^4), lr)$
3	2	-3	slice$(lr/2/(2.6^4), lr/2)$
4	1	—	slice$(lr/10/(2.6^4), lr/10)$
5	1	—	slice$(lr/10/(2.6^4), lr/10)$

Table 5.3: The process of training a target classifier. For each one-cycle policy-based training step, a number of epochs performed is provided, along with an index of layer that the model is unfrozen to (where — means that the entire model is unfrozen, -1 means that the final layer is unfrozen, -2 means that final layer and the preceeding layer is unfrozen, etc.) and a `max_lr` value passed to the `fit_one_cycle` method of `fast.ai` library.

5.10 Summary of contributions

While a resulting target classifier model has been produced based on ULMFiT method, several important contributions are made in this work in order to produce the target document-level sentiment analysis system:

- A design of the resulting system is proposed (see section 5.3) based on defined functional and non-functional requirements (see section 5.2)
- A `plwiki` dataset, based on the entire contents of Polish Wikipedia, is produced in order to pre-train a domain-general language model (see section 5.5.2),
- A `Filmweb+` dataset, containing reviews fetched from major Polish movie review blogs and websites, is produced in order to fine-tune a domain-specific language model (see section 5.5.1)
- Based on prior research, a `sentencepiece`-based tokenization is applied in order to produce an efficient token vocabulary for a highly inflected Polish language (see section 5.6.1),
- Hyperparameters and training strategies are chosen based on performed experiments (see chapter 7) in order to obtain a state-of-the-art comparable document sentiment classification performance,
- System's functional and non-functional requirements (see section 5.2) are defined, resulting in an proposed design and implementation (see section 5.3).

Chapter 6

User manual

This chapter presents a user manual for a sentiment classification system. It also contains requirements needed to be met in order for a sentiment classification system to run on a computer system.

6.1 System requirements

The sentiment classification system requires all of the hardware and software requirements stated below to be met in order for it to work properly.

6.1.1 Hardware requirements

In order to run a sentiment classification system, a computer system should have:

1. `Nvidia GeForce 1080Ti` graphics card with least `11 GB` of `GDDR6 VRAM`,
2. `16 GB` of available system memory,
3. `20 GB` of available hard disk memory.

An usage of `Nvidia GeForce 2080Ti` graphics card is recommended for an increase of training-related computations' speed.

6.1.2 Software requirements

A computer system should run on a `Ubuntu 18.04` or `macOS 10.15.1 Catalina` operating system. Operating system should have:

1. `CUDA` drivers compatible with a graphics card and an operating system installed,
2. `Python 3.7.5` with `pip 19.3.1` installed.

`Python 3.7.5` should be run upon typing a `python` command in a system terminal. In order to verify that the proper version of Python interpreter is running, run a `python --version` command.

6.1.3 Software dependencies

The following `Python` packages are needed to be installed in order for an sentiment classification system to work properly:

1. `Pandas` data processing library version `0.25.3`, used for pre-processing of raw `Filmweb+ data`,
2. `fast.ai` deep learning library version `1.0.58`, used for creating `DataBunch` objects and training models,
3. `sentencepiece` tokenization library version `0.1.85`, used for tokenization,
4. `PyTorch` machine learning library version `1.3.1`, used as a `fast.ai` backend used for performing training-related computations.

Aforementioned packages' subdepedencies are also required.

Optionally, the software dependencies can be managed using `pipenv`, which can installed by `pip`. After `pipenv` is successfully installed, running `pipenv shell` command in the main project directory followed by `pipenv install` command will result in a successful installation of all required packages.

6.2 Command-line interface

The command-line interface of a sentiment classification system enables its user to perform a classification of input text documents or to train custom classifier model used for latter predictions. This section describes an usage of three distinctive modes of the CLI: the *classification* mode, the *data preparation* mode and the `training` mode.

6.2.1 Classification mode

The classification mode (`classify`) allows for a document-level sentiment classification to be performed either on a single document or multiple documents stored in `.txt` format in a given `path`.

Parameters available in the classification mode:

- `--models_path`, which determines the path to the prepared model files (defaults to `~/data/models/`),
- `--uncertainty_level`, which defines a maximum value of an absolute difference between $|p_p - p_n|$, where p_p is a computed probability of a document belonging to a positive class, and p_n is a probability of a document belonging to a negative class, for which the model outputs that the document belongs to a neutral class (defaults to `5`),

Based on a given input, the system produces an output containing a path to a document, assigned label (positive, neutral or negative), and calculated probabilities of a document belonging to a positive and a negative class. Input documents need to meet the requirements defined in section 2.3 in order for a valid classification to be produced.

6.2.2 Data preparation mode

In a data preparation mode (`prepare_data`) the sentiment classification system generates data used for a latter training of language models and target classifiers.

Parameters available in the data preparation mode:

- `--wikipedia_path`, which defines a path where the contents of Wikipedia is downloaded to (defaults to `~/data/wikipedia/`),
- `--filmwebplus_path`, which defines a path where the `filmwebplus.csv` file is stored at (defaults to `~/data/filmwebplus/filmwebplus.csv`),
- `--language_code`, which defines a code of a language (e.g. `pl, en`) for which the contents of the Wikipedia is downloaded (defaults to `~/data/wikipedia/`),
- `--min_article_len`, which defines a minimum length of an Wikipedia article to be processed and put in a dataset (defaults to `2600`),
- `--batch_size`, which defines a size of a single batch of data used in a latter training process (defaults to `64`),
- `--vocab_size`, which defines a token vocabulary size to use (defaults to `32000`),
- `--cpu_count`, which defines a number of CPUs to use during `DataBunch` processing (defaults to `8`),
- `--databunches_path`, which defines a path to where the created databunches are stored (defaults to `~/data/databunches/`).

When the contents of a Wikipedia for a new language needs to be downloaded and pre-processed, the system relies on a `wikiextractor` library, used for an extraction of plain text from downloaded raw Wikipedia contents[1].

[1]https://github.com/attardi/wikiextractor

As a result of running the system in the data preparation mode, three separate databunches are generated: a domain-general databunch, based on downloaded Wikipedia data, used to train the domain-general language model, a domain-specific databunch, based on unlabelled Filmweb+ dataset contents, used to train the domain-specific language model, and a target classifier databunch, based on labelled Filmweb+ dataset contents, used to train the target classifier.

6.2.3 Training mode

The document sentiment classification system's training mode (`train`) is used to perform a training of language models and target classifiers based on databunches generated in a data preparation mode (see section 6.2.2).

Parameters available in the training mode:

- `--databunches_path`, which defines a path to where the databunches created in a data preparation mode are stored (defaults to `~/data/databunches/`),
- `--models_path`, which defines a path to where the created language models and a target classifier model are stored (defaults to `~/data/models/`),
- `--batch_size`, which defines a size of a single batch of data used in training (defaults to `64`),
- `--general_lm_epochs`, which defines an amount of training epochs used when training an entirely unfrozen domain-general language model (defaults to `10`),
- `--ds_lm_frozen_epochs`, which defines an amount of training epochs used when training a domain-specific language model's final layer (defaults to `2`),
- `--ds_lm_epochs`, which defines an amount of training epochs used when training an entirely unfrozen domain-specific language model (defaults to `8`),
- `--target_max_lr`, which defines a maximum learning rate used when training a target classifier (defaults to `2e-2`),
- `--target_step[x]_epochs`, which defines an amount of epochs per single training instance application `[x]` (where $\texttt{x} \in \{1, 2, \dots, 5\}$) performed in a final classifier training (defaults to a value of `2` for $\texttt{x} \in \{1, 2, 3\}$, and `1` for $\texttt{x} \in \{4, 5\}$).

As a result of running the system in a training mode, three separate models are produced: a domain-general language model, a domain-specific language model, and a target classifier.

6.3 Usage

In order to use a document-level sentiment classification system in any of its modes using a desired combination of parameters, run `python main.py [mode] --argument value` command in a system terminal. In place of `[mode]`, use either `classify`, `prepare_data` or `train`, and instead of `--argument value` provide any of the aforementioned arguments with a corresponding value.

6.3.1 First run

It is required to generate training data and train a target classifier model first before predictions can be made using the document sentiment classification system.

In order to do so, first download a `filmwebplus.csv` file from a remote server using the `download.sh` script located in a main directory of the document sentiment classification system, `fincher`, and then run `python main.py prepare_data` command, followed by `python main.py train` (providing optional argument values if deemed necessary).

Chapter 7

Experiments

Having discussed the design and an implementation of the sentiment classification system in chapter 5, this chapter presents the results of experiments performed on a experimental set-up in order to determine the impact of the hyperparameters, the size of the token vocabulary and an amount of training data used on the performance of an underlying machine learning model.

7.1 Experimental set-up

7.1.1 Environment

The experiments have been performed on `Vast.ai`[1] cloud instances equipped with a lstinline11.2 GB of GPU memory-equipped `Nvidia RTX 2080Ti` graphics card, running a `vastai/pytorch Docker` image[2] using `CUDA 10 Turing` architecture-compatible drivers.

7.1.2 Global assumptions and parameters

In the following experiments, a single training instance is always performed using a one-cycle policy if not explicitly stated otherwise.

Fixed hyperparameters referring to the momentum used during the training process, the `mom_min` value of 0.7 and `mom_max` value of 0.8 are used, and a constant `batch_size` of 64 is used. The parameters involved with setting the learning rate during training (see section 5.9.2), an `max_lr` value and a `lr` constant, are always scaled by a factor of `batch_size/24`

The `plwiki` dataset is identically split into a training set and a validation set for each of the tested domain-general language models, with a validation set consisting of 10% of total amount of data available. `Filmweb+` dataset is similarly split for domain-specific language models.

For classifiers, we identically split `Filmweb+` into training set (which contains 80% of all available data), validation set (10% of all available data) and test set (10% of all available data) for each model used in an experiment. The validation set is used to compare performance between different models, and the test set - to measure the performance of a chosen model on data unseen during training.

A *flattened cross entropy loss* function is used in the experiments.

7.2 Token vocabulary size and target classifier performance

We perform several experiments in order to find an appropriate token vocabulary V^* size to use with a resulting final classifier. For this purpose, we pre-train and fine-tune four different language models which are then used to produce a final classifier, on data tokenized using the `unigram` model-based `sentencepiece` tokenizer with $|V^*| \in \{20000, 24000, 28000, 32000\}$.

[1]https://vast.ai/
[2]https://hub.docker.com/r/vastai/pytorch/

7.2.1 Accuracy of domain-general language models

We first measure an impact of the amount of tokens in a vocabulary V^* on the accuracy of a domain-general language model. For this purpose, four domain-general language models are pre-trained on tokenized `plwiki` datasets for 10 epochs, with maximum learning rate `max_lr` set according to table 7.1.

Pre-trained language models		Fine-tuned language models	
Vocabulary size	Maximum learning rate	Vocabulary size	Maximum learning rate
20000	$1.74 \cdot 10^{-3}$	20000	$6.31 \cdot 10^{-7}$
24000	$2.09 \cdot 10^{-3}$	24000	$6.31 \cdot 10^{-7}$
28000	$1.74 \cdot 10^{-3}$	28000	$1.10 \cdot 10^{-7}$
32000	$2.09 \cdot 10^{-3}$	32000	$3.02 \cdot 10^{-5}$

Table 7.1: A `max_lr` value chosen for pre-training (on the left) and fine-tuning (on the right) a language model depending on a size of the vocabulary. The values were determined using `fast.ai lr_finder`.

It is observed that the size of vocabulary has an impact on the accuracy of a domain-general LM on a validation set. Increasing the amount of tokens in the vocabulary results in an increase of the general language model's loss observed on a validation dataset, as shown in figure 7.1.

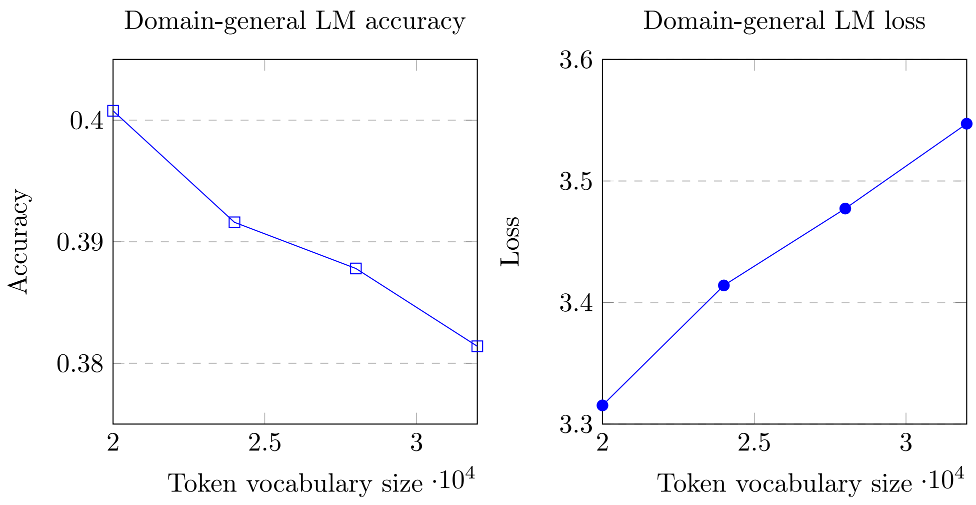

Figure 7.1: A general language model accuracy (on the left) and loss (on the right) measured on a validation set in relation to the size of the token vocabulary.

7.2.2 Accuracy of domain-specific language models

Similarly to domain-general language models, we measure an impact of vocabulary size used on the domain-specific language model by fine-tuning four previously produced language models.

The fine-tuning begins with a training of a final layer of the model for 2 epochs. After that, an entire language model is unfrozen, and additional fine-tuning is performed for 8 more epochs. For the entire fine-tuning process, `max_lr` is set according to table 7.1. The results are shown in a figure 7.2.

Most notably, the accuracy of fine-tuned language models on a validation sets first decreases when a size of the token vocabulary increases, only to increase again when a vocabulary of size 32000 is chosen.

While the language models trained on a vocabulary of size 20000 and 24000 share the exact same `max_lr` rate and other hyperparameters, the performance of a 24000-token vocabulary language model is significantly lower than that of a 20000-token vocab one. The 32000-token vocab LM performs substantially better among the others, and the 28000-vocab LM - the worst.

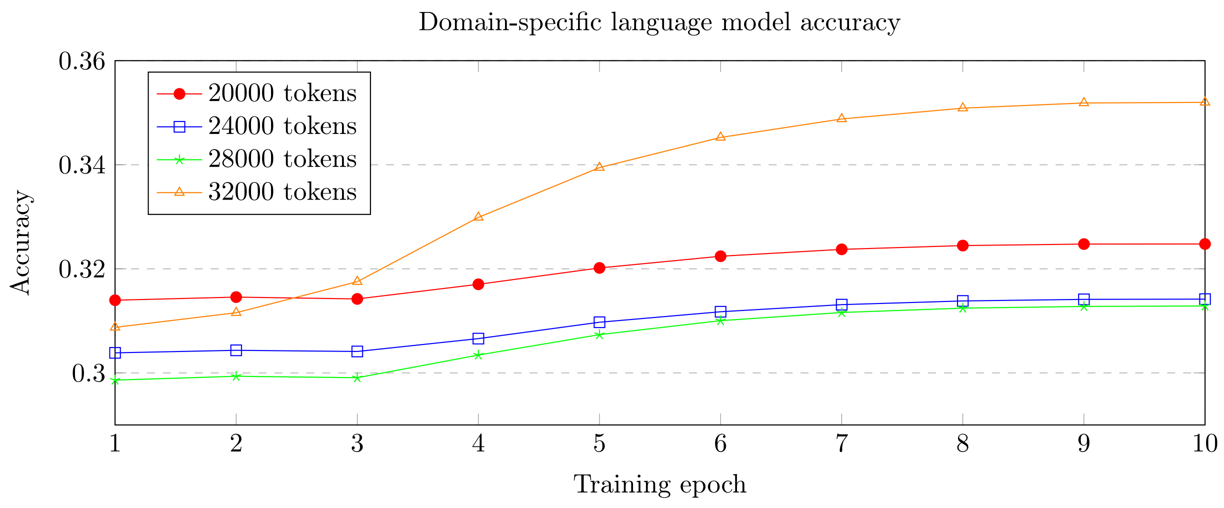

Figure 7.2: A plot of domain-specific language model accuracy on a validation set depending on the size of the token vocabulary.

Target classifier training process				Unfrozen classifier training process		
Step	Epochs	Frozen to layer	Maximum learning rate	$\|V^*\|$	Epochs	Steps
1	2	-1	lr	20000	1	2
2	2	-1	lr	24000	2	1
3	2	-2	slice$(lr/(2.6^4), lr)$	28000	1	1
4	2	-3	slice$(lr/2/(2.6^4), lr/2)$	32000	2	1

Table 7.2: A target classifier training process, which is performed in order to test the impact of token vocabulary size on the accuracy of the model. As a final step, final fine-tuning of a target classifier is performed depending on $|V^*|$ as shown in a rightmost table, with a `max_lr` value of `slice(lr/10/(2.6^4), lr/10)`.

7.2.3 Accuracy of target classifiers

Finally, a relation between an accuracy of target classifiers and the token vocabulary size is measured. Target classifiers are gradually unfreezed and trained starting from the final, linear decoding layers. A constant `lr` value of $2 \cdot 10^{-2}$ is used during this process.

For each model, the training process differs only in the training performed on an entirely unfrozen classifier model. Training steps utilising gradual unfreezing are shown in table 7.2, and the final classifier fine-tuning, performed to maximise the models' accuracy while avoiding an occurrence of overfitting, is shown in table 7.2.

We first compare the models' accuracy (plot 7.3) on a validation set, as well as loss on both training and a validation sets (plot 7.4) in relation to the number of training epochs performed. The results indicate that a model using the largest token vocabulary V^* available consistently outperforms all of the other models operating on smaller $|V^*|$. For this model, a comparison of training and validation loss shown on plot 7.4 leads to a conclusion that this model is not overfitting on a validation set. What's more, each model reduces its initial loss on both sets of data in the final training epoch, achieving similiar, high accuracy of over 90% on a validation set.

A comparison of models' performance is also done using a test set of labelled text document examples on trained target classifiers. A plot of relation between model's accuracy and loss on a test set 7.5 indicates that a larger amount of tokens in V^* results in an increase of models' accuracy, with a 28000-token vocabulary-based model performing worse than expected.

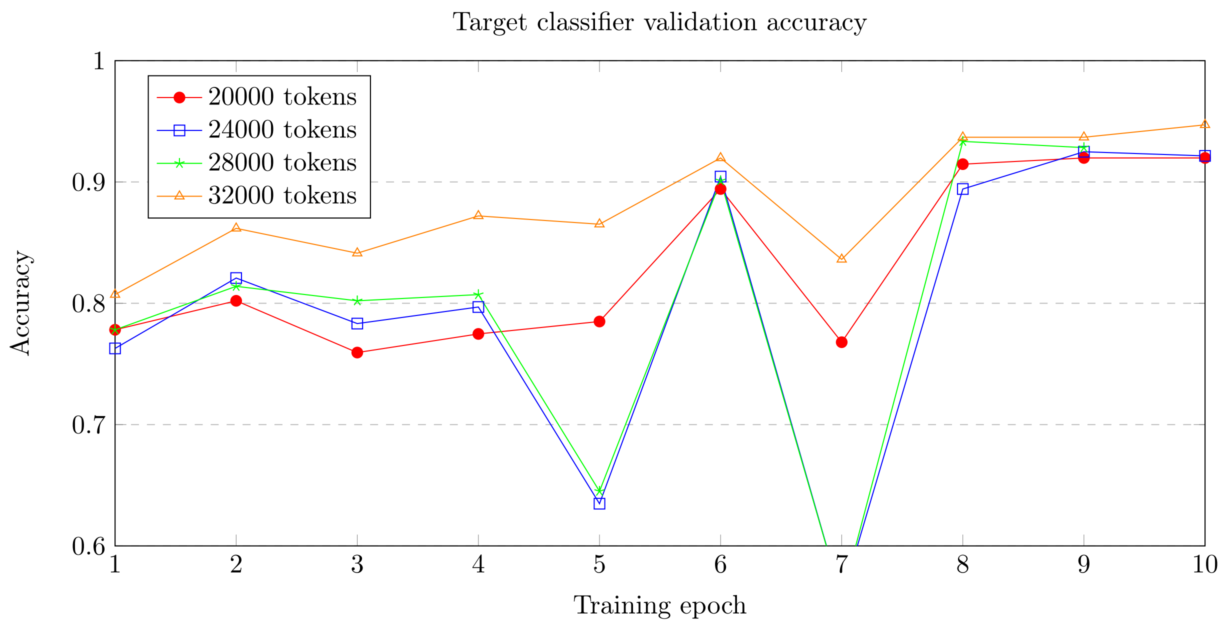

Figure 7.3: A plot of target classifiers' accuracy on a validation set depending on the size of the token vocabulary. The 32000-token vocab classifier produces the highest validation accuracy in each training epoch. In epoch 7, the 28000-token vocab classifier performance drops to 0.539249 accuracy.

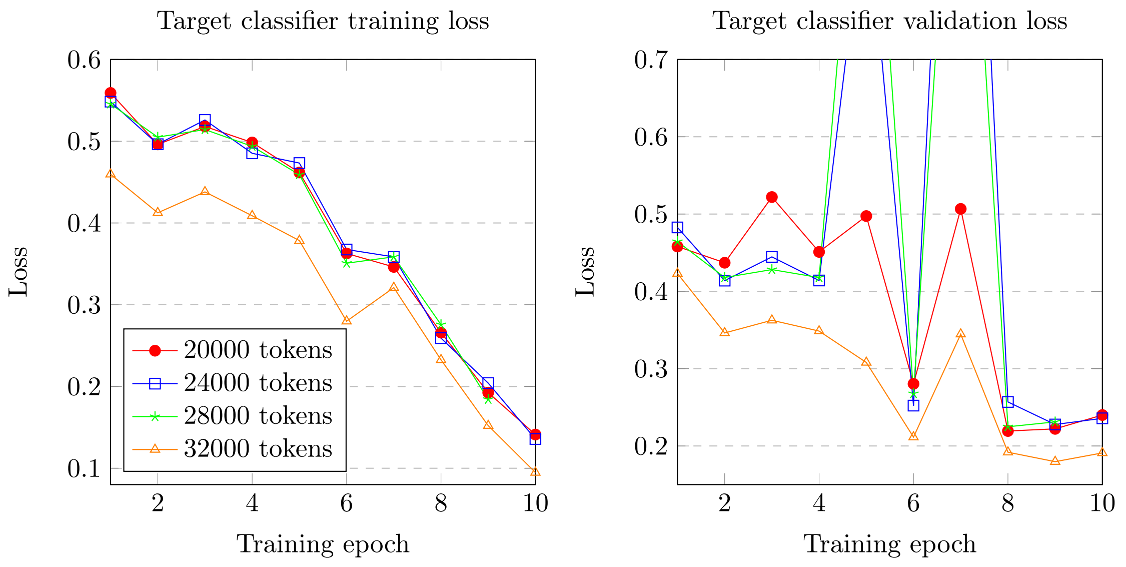

Figure 7.4: A comparison of classifiers' loss on a training and a validation set. The 24000-token vocab classifier loss peaks at 1.497791 on a validation set, and the 28000-token vocab classifier - at 1.203454.

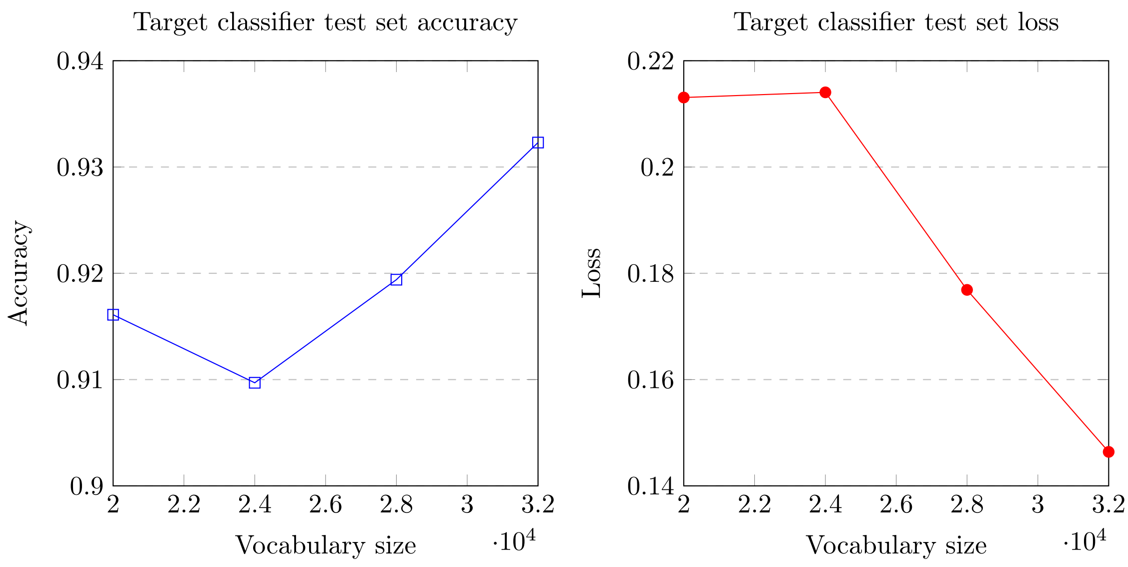

Figure 7.5: A comparison of target classifiers' accuracy (on the left) and loss (on the right) measured on a test set in relation to the size of the token vocabulary.

7.2.4 Conclusions and remarks

While a comparison of domain-general language models' accuracy and loss (section 7.2.1) finds that an increase of $|V^*|$ might not be beneficial, the test results performed on domain-specific language models (section 7.2.2) and target classifiers (section 7.2.3) prove otherwise.

This leads to a conclusion that a larger token vocabulary leads to a better performance of language models and classifiers operating on domain-specific data, which is often more nuanced than general, encyclopedic information a domain-general language model is trained on.

However, a too high a value of $|V^*|$ may lead to a high perplexity of language models, thus decreasing the overall performance, as indicated by previous research [16]. For this reason, and due to the computational difficulty of pre-training a domain-general language model, an impact of $|V^*|$ was measured only in a reasonable range of tokens.

Worse than expected performance of a 24000-token vocabulary-based domain-specific language model, as well as the 24000-token vocab target classifier (section 7.2.2) indicates an importance of careful fine-tuning of models' hyperparameters.

7.3 Labelled data amount and target classifier performance

In order to measure how an increase of available labelled data affects the target classifier's performance, we train several classifiers using subsets L_i of examples from `Filmweb+` labelled dataset. For each model, a subset of size $|L_i| \in \{500, 1000, 1500, 2000, 2500\}$ is chosen and used during training. Each subset L_{i+1} extends contents of L_i with additional 500 labelled examples from entire dataset, that is $L_1 \subseteq L_2, L_2 \subseteq L_3, \ldots, L_4 \subseteq L_5$. Furthermore, each subset contains equal amount of positive and negative examples.

7.3.1 Target classifiers training

The classifiers are gradually unfrozen and trained as described in a table 7.2. The final-tuning training step performed on an entirely unfrozen model is identical for each of the tested models, and consists of a single, one-epoch training step with a `max_lr` set to `slice(lr/10/(2.6^4), lr/10)`, with an exception of the $|L_i| \in \{2000, 2500\}$, trained for two epochs in a single, final training step.

7.3.2 Results and conclusions

The performance of trained target classifiers' is measured on a test set, and the model's loss and accuracy depending on the size of a labelled dataset used for training is shown on plot 7.6.

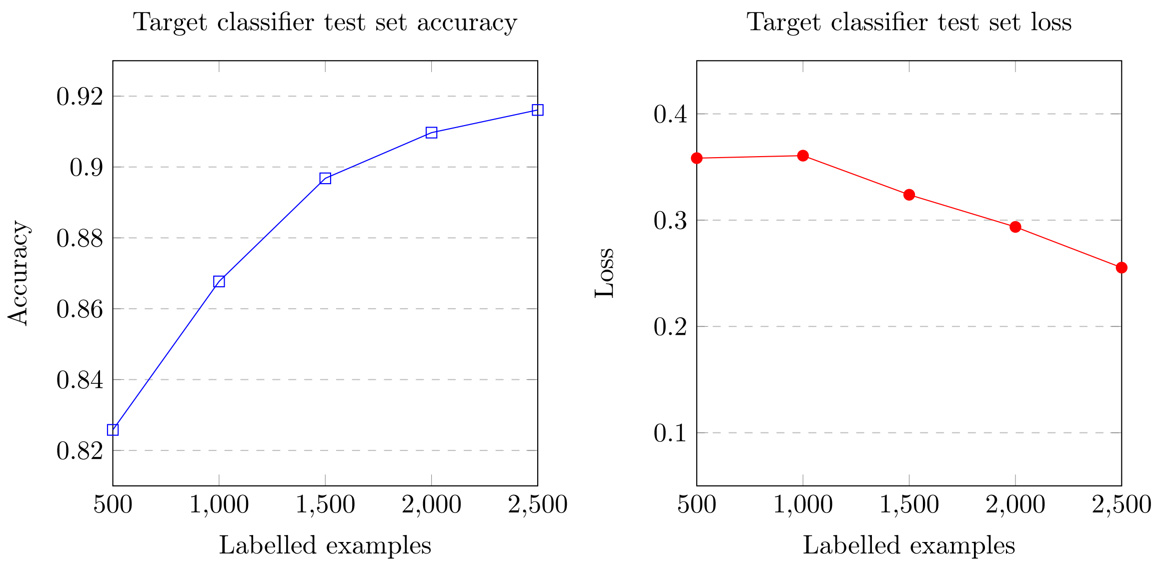

Figure 7.6: A comparison of target classifiers' accuracy (on the left) and loss (on the right) measured on a test set in relation to the size of the labelled dataset available.

An increase of an amount of labelled training examples used for training directly impacts the accuracy of a target classifier, which increases with an corresponding increase of the dataset size. The accuracy curve shown on plot 7.6 indicates that an increase of labelled datasets' size has the largest effect when small datasets ($|L_i| \in \{500, 1000\}$) are used, and its impact gradually diminishes as the larger $|L_i|$ is available.

The loss of a classifier on a test set gradually increases the more labelled training examples there are available, with an exception of first two steps, where for $|L_i| \in \{500, 1000\}$ models' overfitting to a training set is observed. The more labelled training examples are available, the better resulting models' generalise.

Notably, having just 2000 labelled training examples available, the resulting classifier is able to produce over 90% of accuracy on a test set.

In conclusion, an increase of an amount of labelled dataset's size has a positive impact on the performance of a target classifier, especially, when a limited amount is available to begin with.

7.4 Model training time

In order to determine the amount of time needed to produce a target classifier, we perform measurements of single epoch training time of domain-general language models, domain-specific language models, and target classifiers defined in section 7.2 for a token vocabulary of size $|V^*| \in \{24000, 28000\}$.

Measurements are performed on a `Vast.ai` instance equipped with a single `GeForce RTX 2080 Ti` graphics card (see 7.1.1), `WS X299 SAGE` motherboard, `Intel Core i7-9800X` processor and `Samsung` SSD drive with available disk bandwith of `546 MB/s`, producing `18.3` teraflops of GPU performance, and rated `19.1` according to `Vast.ai`'s deep learning performance score, `DLPerf`.

Models using different V^* are compared in order to determine an impact of $|V^*|$ on training time.

7.4.1 Results and conclusions

The target classifier using a token vocabulary V_1^* of 28000 tokens is obtained in just 4 minutes and 28 seconds more as compared to a model with $|V_2^*| = 24000$. Domain-specific language model using V_1^* is obtained in 3 hours and 37 seconds more than a V_2^* model, and, for the same V_2^*, target classifier is obtained 1 minute and 18 seconds faster than one for a smaller V_1^*. However, excluding the additional training step for smaller V_1^*, the difference amounts to just 22 seconds.

In conclusion, the target classifier can be obtained in less than 10 hours when using widely available, consumer-grade hardware. An use of more powerful graphic cards, such as `Nvidia Tesla V100`, or multiple graphic cards for computations may result in a further decrease of total training time.

An impact of an increase of $|V^*|$ by 4000 tokens on the training time is negligible, amounting to just 4 minutes and 28 seconds in over 9-hours-long process. However, larger token vocabularies (such as $|V^*| = 50000$) might have a more significant impact on the overall training process duration.

7.5 Training epochs, iterations and target classifier performance

We determine a significance of an amount of epochs chosen during the target classifiers' training for their validation and test set performance by training three separate models with 2, 4 and 8 epochs in each separate training instance (see test assumptions 7.1.2). Based on the results from this test, a model producing the best overall performance is chosen for latter study.

We also measure an impact of modifying training strategies (see table 7.2) applied for the best performing model's first and final layers, increasing or decreasing an amount of training epochs for chosen training instances, or adding another separate training steps.

7.5.1 Results

Global training epochs amount change and classifier performance

Models' performance measured on a validation set (plot 7.7) and test set (plot 7.8) leads to a conclusion that training a final classifier using the one-cycle policy with two epochs of iteration on the entire training set contents results in a model with the best overall performance.

While an increase of training epochs amount leads to a higher validation accuracy when 4 training epochs are performed, it also results in a model overfitting to a training set, which negatively impacts it's generalisation ability. Furthermore, a further increase to 8 training epochs results in an increase of model's loss on a validation set.

Based on these findings, the model trained with 2 epochs per training step is determined to be the best performing.

Modification of training strategy for final model layer and an entirely unfrozen model

Having determined the best performing model's training strategy we measure an impact of additional tweaks on the overall classification performance.

An increase from 2 to 8 of epochs in the first training step for the best performing model results in a slight increase of its validation set accuracy (from 0.9164 to 0.9198), with a decrease of the test set loss (from 0.2539087 to 0.22110389) and unchanged test accuracy of 0.9226.

A further decrease of epochs for the final training step from 2 to 1 leads to a decrease of validation accuracy (to 0.9061) and test loss (to 0.20510076) and an increase of test accuracy (to 0.9258) and validation loss (0.24217011).

A final addition of 1-epoch training step with unchanged learning rate or other hyperparameters, as compared to the previous final step (see table 7.2) results in an additional test accuracy, test loss and validation loss increase (to 0.9323, 0.22691214 and 0.28758347 respectively) and a validation accuracy decrease (to 0.9044).

These trials lead to an observation that an increase of training amount performed on the final layer of the model alone has a negligible impact on the model performance and, taking into account additional computational costs, can not be performed. A change in the strategy performed on an entirely unfrozen model, on the other hand, leads to a more significant difference in the model's performance.

For this reason, another modification to the training strategy (see table 7.2) is tested:

1. A decrease of training instances performed on the final model layer from 2 to 1,
2. A decrease of epochs in a final step from 2 to 1,
3. An addition of final, one-epoch training step, without a change of learning rate or other hyperparameters as compared to the previously final step.

This change results in an unchanged validation set accuracy (0.9164), but a noticeably smaller validation set loss (change from 0.2539087 to 0.23734723), higher test set accuracy (change from 0.9226 to 0.9419) and lower test set loss (change from 0.20687753 to 0.20117973). This change leads to a model achieving a better generalisation performance for new, previously unseen data, as indicated by an increase of test set accuracy.

7.5.2 Conclusions

We have found that further modifications of the training strategy described in table 7.2 result in an increase of the overall target classifiers' performance.

While the tweaks performed on first training steps of a final classifier did not result in a significant difference, changes applied to a training policy for an entirely unfrozen model allowed for a decrease of validation and test set loss, resulting in over 94% of model's test set accuracy, possible due to an improvement of the generalisation ability to new, unseen data.

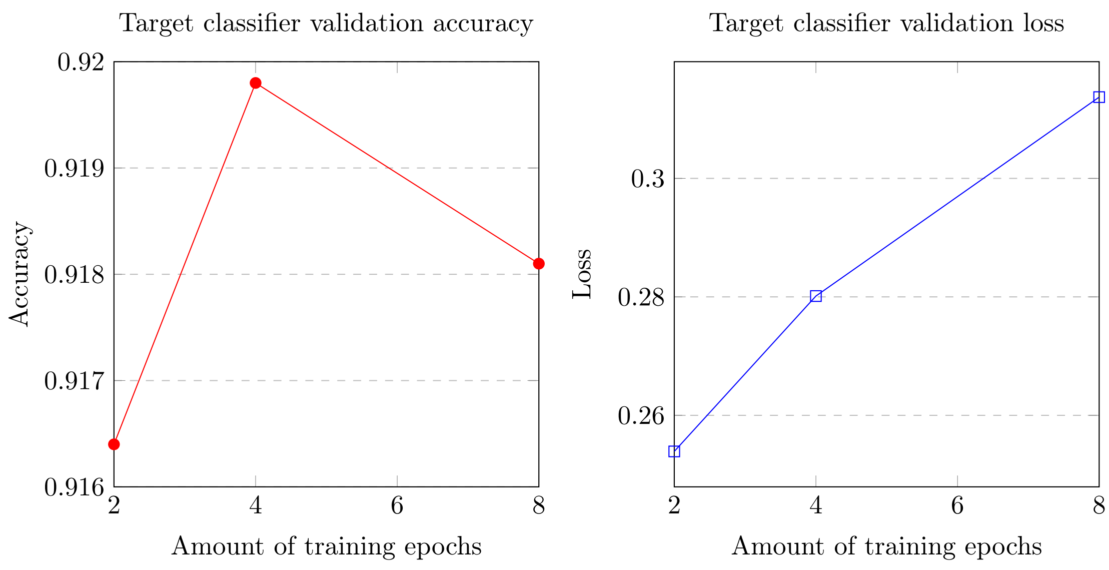

Figure 7.7: A comparison of target classifiers' accuracy (on the left) and validation loss (on the right) in relation to the amount of epochs per single unique training instance used.

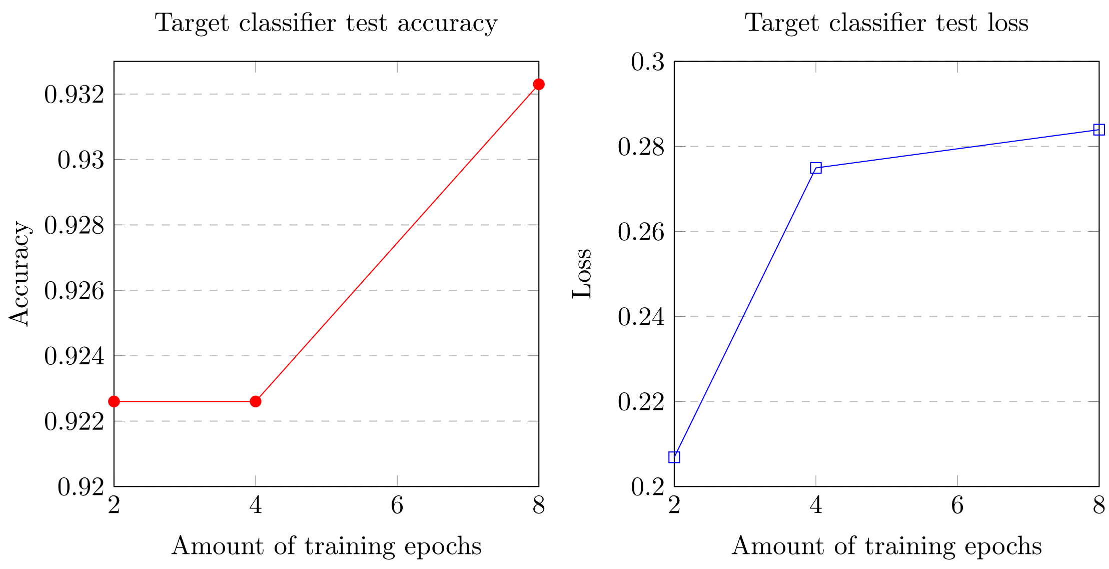

Figure 7.8: A comparison of target classifiers' accuracy (on the left) and validation loss (on the right) in relation to the amount of epochs per single unique training instance used.

Training time measurements			
Model	Epoch	$\|V^*\|$ 24000	$\|V^*\|$ 28000
Domain-general language model	1	45:08	45:34
	2	45:05	45:18
	3	45:05	45:18
	4	44:48	45:18
	5	44:53	45:16
	6	44:49	45:04
	7	44:51	45:07
	8	45:02	45:19
	9	45:01	45:17
	10	45:03	44:53
		7:29:45	7:32:24
Domain-specific language model	1	08:21	08:48
	2	08:32	09:03
	3	10:25	10:42
	4	10:26	10:46
	5	10:26	10:44
	6	10:22	10:44
	7	10:23	10:43
	8	10:25	10:45
	9	10:24	10:43
	10	10:25	10:44
		1:40:20	1:43:57
Target classifier	1	00:31	00:26
	2	00:31	00:27
	3	00:33	00:28
	4	00:32	00:28
	5	00:38	00:32
	6	00:41	00:34
	7	00:52	00:44
	8	00:49	00:43
	9	00:54	00:51
	10	00:56	-
		0:06:57	0:05:13
Total training time		9:16:51	9:21:19

Table 7.3: Total training time of domain-general language models, domain-specific language models and target classifiers with $|V^*| \in \{24000, 28000\}$ per training epoch.

Chapter 8

Final remarks

The following chapter concludes the work on a sentiment classification system, re-iterating achieved results from the work on the system, as well as key insights. It also states future directions to take in order to further explore the problem of neural networks-based document sentiment classification.

8.1 Summary of work

We have defined a problem of a document sentiment classification and provided the reader with a comprehensive background on recurrent neural network-based natural language processing, starting from important machine learning problems and ending with an perspective on impact of recent developments in the deep learning on the field of NLP.

Then, we have presented a proposed implementation of a ULMFiT-based Polish text documents sentiment classification system, based on `plwiki` and `Filmweb+` datasets, introducing various pre-training and fine-tuning strategies, regularisation and tokenization techniques, and manually selected hyperparameter values.

We have also provided a manual for a resulting document-level sentiment classification system, presenting an overview system's requirements, as well as usage of an available command-line interface for the purpose of performing an input document sentiment classification and training new classifier models.

Finally, in order to test an impact of different factors of an underlying machine learning model, we have performed and documented several experiments on a token vocabulary size, amount of labelled training data used and hyperparameters.

8.2 Conclusions

In conclusion, the main goal of this work, the design an implementation of a document-level sentiment classification system on a practical, real-world domain of Polish movie reviews has been successfully accomplished.

In fact, the proposed implementation achieves a test set accuracy of 94,19 %, making it a state-of-the-art result for a real-world, long text (500 words on average) document sentiment analysis task in Polish language [69, 16, 11], second only to performance of neural network-based implementations performing classification on short product opinions [95, 2].

8.3 Future directions

Having proven that it is possible to achieve state-of-the-art comparable performance on a Polish document-level sentiment classification task using inductive transfer learning methods on deep recurrent neural network-based classifiers produced from pre-trained and fine-tuned language models, an impact of several latest developments in the fields of NLP and deep learning on the target classification performance remains to be measured:

- Utilization of *quasi-recurrent neural networks* as an underlying neural network infrastructure,
- Application of *subword regularization* during model training,
- Usage of *ensemble models* as a mean of improving accuracy,
- Implementation of *cross-lingual* and *multi-lingual* language models [21],
- Variable amount of unlabelled training examples used to train domain-general and domain-specific language models.

The resulting document-level sentiment classification system can be already used as a base for a commercial product used to measure the Polish audience's response towards movies or television series produced by media companies, based on reviews published on blogs and websites.

However, due to the system's modular nature being a consequence of applied inductive transfer learning approach, it can be easily applied for a problem of document-level sentiment classification in other domains of sentiment analysis, including, but not limited to political sentiment, brand sentiment, or product sentiment. This can be done for documents in a Polish language, leveraging already pre-trained domain-general language model available, by producing a model capable of working in any other language of choice.

Finally, the resulting target classifier can be applied not only for a problem of a document-level sentiment analysis, but also for any feasible text classification task of choice, such as document topic classification, spam recognition, or profanity detection. For instance, in a domain of market research, possible applications include:

- Detection of automatically-generated reviews by document-level binary classification (written by human, generated by software) or pattern-based classification (unusual flattery, odd formatting, etc.) used in competition research to discard unwanted opinions,
- Verification of output from a generator of popular topics, products or articles used to fit the market demand,
- Grouping of feedback on the product based on its features, such as price, design, or functionalities, for the purpose of future improvements.

Bibliography

[1] David Bachman. *Advanced Calculus Demystified.* McGraw-Hill Professional, 2007. ISBN: 0071481214. DOI: 10.1036/9780071511094. URL: https://mhebooklibrary.com/doi/book/10.1036/9780071511094.

[2] Roman Bartusiak et al. “Sentiment Analysis for Polish Using Transfer Learning Approach”. In: Sept. 2015. DOI: 10.1109/ENIC.2015.16.

[3] Dan Becker. *Rectified Linear Units (ReLU) in Deep Learning.* 2017. URL: https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning (visited on 11/28/2019).

[4] Yoshua Bengio, Aaron Courville, and Pascal Vincent. *Representation Learning: A Review and New Perspectives.* 2012. arXiv: 1206.5538 [cs.LG].

[5] John Blitzer, Mark Dredze, and Fernando Pereira. “Biographies, Bollywood, Boom-boxes and Blenders: Domain Adaptation for Sentiment Classification”. In: *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics.* Prague, Czech Republic: Association for Computational Linguistics, June 2007, pp. 440–447. URL: https://www.aclweb.org/anthology/P07-1056.

[6] Tomasz Boiński and Adam Chojnowski. “Towards facts extraction from text in Polish language”. In: July 2017, pp. 13–17. DOI: 10.1109/INISTA.2017.8001124.

[7] Aleksander Buczynski and Aleksander Wawer. “Shallow parsing in sentiment analysis of product reviews”. In: *Proceedings of the Partial Parsing workshop at LREC.* Vol. 2008. 2008, pp. 14–18.

[8] Eric Cai. *Machine Learning Lesson of the Day – Overfitting and Underfitting.* 2014. URL: https://chemicalstatistician.wordpress.com/2014/03/19/machine-learning-lesson-of-the-day-overfitting-and-underfitting (visited on 12/15/2019).

[9] Kevin Casey. *How to explain natural language processing (NLP) in plain English.* 2019. URL: https://enterprisersproject.com/article/2019/9/natural-language-processing-nlp-explained-plain-english (visited on 11/23/2019).

[10] Arjun Chaudhuri. “Chapter 2 - Emotion and Reason”. In: *Emotion and Reason in Consumer Behavior.* Ed. by Arjun Chaudhuri. Boston: Butterworth-Heinemann, 2006, pp. 25–38. ISBN: 978-0-7506-7976-3. DOI: https://doi.org/10.1016/B978-0-7506-7976-3.50006-3. URL: http://www.sciencedirect.com/science/article/pii/B9780750679763500063.

[11] Karol Chlasta. “Sentiment analysis model for Twitter data in Polish language”. PhD thesis. June 2015.

[12] François Chollet. *Deep Learning with Python.* Manning, Nov. 2017. ISBN: 9781617294433.

[13] Fast.ai Contributors. *Fast.ai Documentation.* 2019. URL: https://docs.fast.ai/text.transform.html (visited on 12/20/2019).

[14] Khan Academy contributors. *The gradient vector.* URL: https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/partial-derivative-and-gradient-articles/a/the-gradient (visited on 12/05/2019).

[15] TensorFlow Documentation Contributors. *Word embeddings.* 2019. URL: https://www.tensorflow.org/tutorials/text/word_embeddings (visited on 12/09/2019).

[16] Piotr Czapla, Jeremy Howard, and Marcin Kardas. *Universal Language Model Fine-Tuning with Subword Tokenization for Polish.* 2018. arXiv: 1810.10222 [cs.CL].

[17] Andrew M. Dai and Quoc V. Le. *Semi-supervised Sequence Learning.* 2015. arXiv: 1511.01432 [cs.LG].

[18] J. Deng et al. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *CVPR09*. 2009.

[19] scikit-learn developers. *Underfitting vs. Overfitting*. 2014. URL: https://scikit-learn.org/0.15/auto_examples/plot_underfitting_overfitting.html (visited on 12/02/2019).

[20] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *NAACL-HLT*. 2019.

[21] Julian Eisenschlos et al. *MultiFiT: Efficient Multi-lingual Language Model Fine-tuning*. 2019. arXiv: 1909.04761 [cs.CL].

[22] Dumitru Erhan et al. “Why Does Unsupervised Pre-training Help Deep Learning?” In: *J. Mach. Learn. Res.* 11 (Mar. 2010), pp. 625–660. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1756006.1756025.

[23] Scott Fortmann-Roe. *Understanding the Bias-Variance Tradeoff.* 2012. URL: http://scott.fortmann-roe.com/docs/BiasVariance.html (visited on 12/15/2019).

[24] Yarin Gal and Zoubin Ghahramani. *A Theoretically Grounded Application of Dropout in Recurrent Neural Networks*. 2015. arXiv: 1512.05287 [stat.ML].

[25] Michael Galveston. *Great book for new scientists or older scientists who can’t write*. 2019. URL: https://www.amazon.com/How-Write-Publish-Scientific-Paper/product-reviews/1440842809 (visited on 12/29/2019).

[26] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. Sebastopol, CA: O’Reilly Media, 2017. ISBN: 978-1491962299.

[27] Yoav Goldberg and Graeme Hirst. *Neural Network Methods in Natural Language Processing*. Morgan & Claypool Publishers, 2017. ISBN: 1627052984, 9781627052986.

[28] Ian J. Goodfellow et al. *An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks*. 2013. arXiv: 1312.6211 [stat.ML].

[29] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[30] *Gradient Descent*. 2017. URL: https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html (visited on 12/01/2019).

[31] Alex Graves. “Generating Sequences With Recurrent Neural Networks”. In: *CoRR* abs/1308.0850 (2013). arXiv: 1308.0850. URL: http://arxiv.org/abs/1308.0850.

[32] Sylvain Gugger. *How Do You Find A Good Learning Rate*. 2018. URL: https://sgugger.github.io/how-do-you-find-a-good-learning-rate.html (visited on 12/22/2019).

[33] Sylvain Gugger. *The 1cycle policy*. 2018. URL: https://sgugger.github.io/the-1cycle-policy.html (visited on 12/22/2019).

[34] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. 2nd ed. Springer, 2009. URL: http://www-stat.stanford.edu/~tibs/ElemStatLearn/.

[35] Simon S. Haykin. *Neural networks and learning machines*. Third. Upper Saddle River, NJ: Pearson Education, 2009.

[36] Olivier J. Hénaff et al. *Data-Efficient Image Recognition with Contrastive Predictive Coding*. 2019. arXiv: 1905.09272 [cs.CV].

[37] Martin Heusel et al. *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. 2017. arXiv: 1706.08500 [cs.LG].

[38] Geoffrey E. Hinton et al. “Improving neural networks by preventing co-adaptation of feature detectors”. In: *CoRR* abs/1207.0580 (2012). cite arxiv:1207.0580. URL: http://arxiv.org/abs/1207.0580.

[39] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-term Memory”. In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[40] Jeremy Howard and Sebastian Ruder. “Universal Language Model Fine-tuning for Text Classification”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 328–339. DOI: 10.18653/v1/P18-1031. URL: https://www.aclweb.org/anthology/P18-1031.

[41] Chip Huyen. *Evaluation Metrics for Language Modeling*. 2019. URL: https://thegradient.pub/understanding-evaluation-metrics-for-language-models (visited on 12/23/2019).

[42] Nitin Indurkhya and Fred J. Damerau. *Handbook of Natural Language Processing*. 2nd. Chapman & Hall/CRC, 2010. ISBN: 9781420085921.

[43] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 `[cs.LG]`.

[44] Rachel Thomas Jeremy Howard. *NLP's ImageNet moment has arrived*. 2018. URL: https://ruder.io/nlp-imagenet (visited on 12/23/2019).

[45] Rachel Thomas Jeremy Howard. *Practical Deep Learning for Coders, v3*. 2018. URL: https://course.fast.ai (visited on 12/22/2019).

[46] Christopher Williams John McGonagle George Shaikouski. *Backpropagation*. 2018. URL: https://brilliant.org/wiki/backpropagation (visited on 12/01/2019).

[47] Yoon Kim. "Convolutional Neural Networks for Sentence Classification". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1746–1751. DOI: 10.3115/v1/D14-1181. URL: https://www.aclweb.org/anthology/D14-1181.

[48] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations* (Dec. 2014).

[49] Will Koehrsen. *Neural Network Embeddings Explained*. 2018. URL: https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526 (visited on 12/09/2019).

[50] Renard Korzeniowski et al. *Exploiting Unsupervised Pre-training and Automated Feature Engineering for Low-resource Hate Speech Detection in Polish*. 2019. arXiv: 1906.09325 `[cs.CL]`.

[51] Kamila Kowalska, Di Cai, and Steve Wade. "Sentiment Analysis of Polish Texts". In: 2012.

[52] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Neural Information Processing Systems* 25 (Jan. 2012). DOI: 10.1145/3065386.

[53] Taku Kudo. "Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 66–75. DOI: 10.18653/v1/P18-1007. URL: https://www.aclweb.org/anthology/P18-1007.

[54] Siwei Lai et al. "Recurrent convolutional neural networks for text classification". In: *Twenty-ninth AAAI conference on artificial intelligence*. 2015.

[55] Zhenzhong Lan et al. *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*. 2019. arXiv: 1909.11942 `[cs.CL]`.

[56] Jey Han Lau and Timothy Baldwin. "An empirical evaluation of doc2vec with practical insights into document embedding generation". In: *arXiv preprint arXiv:1607.05368* (2016).

[57] Bing Liu. *Sentiment Analysis and Opinion Mining*. Morgan & Claypool Publishers, 2012. ISBN: 9781608458844.

[58] Xiaodong Liu et al. *Multi-Task Deep Neural Networks for Natural Language Understanding*. 2019. arXiv: 1901.11504 `[cs.CL]`.

[59] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692 `[cs.CL]`.

[60] Jakub Marian. *Comparison of difficulty of different languages*. 2017. URL: https://jakubmarian.com/are-all-languages-equally-hard-to-learn (visited on 11/23/2019).

[61] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. "Regularizing and Optimizing LSTM Language Models". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=SyyGPP0TZ.

[62] Stephen Merity, Bryan McCann, and Richard Socher. *Revisiting Activation Regularization for Language RNNs*. 2017. arXiv: 1708.01009 `[cs.CL]`.

[63] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems* 26 (Oct. 2013).

[64] Thomas M. Mitchell. *Machine Learning*. 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997. ISBN: 9780070428072.

[65] S MOR-YOSEF. "Ranking the risk factors for cesarean : logistic regression analysis of a nationwide study". In: *Obstet Gynecol* 75 (1990), pp. 944–947. URL: https://ci.nii.ac.jp/naid/10019864417/en/.

[66] Lili Mou et al. *How Transferable are Neural Networks in NLP Applications?* 2016. arXiv: 1603.06111 [cs.CL].

[67] A.C. Müller and S. Guido. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media, 2016. ISBN: 9781449369897. URL: https://books.google.pl/books?id=vbQlDQAAQBAJ.

[68] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2018. URL: http://neuralnetworksanddeeplearning.com/.

[69] Maciej Ogrodniczuk and Łukasz Kobyliński, eds. *Proceedings of the PolEval 2019 Workshop*. Warsaw, Poland: Institute of Computer Science, Polish Academy of Sciences, 2019. ISBN: 978-83-63159-28-3. URL: http://2019.poleval.pl/files/poleval2019.pdf.

[70] Xi Ouyang et al. "Sentiment Analysis Using Convolutional Neural Network". In: Oct. 2015, pp. 2359–2364. DOI: 10.1109/CIT/IUCC/DASC/PICOM.2015.349.

[71] Donghang Pan et al. *Deep neural network-based classification model for Sentiment Analysis*. 2019. arXiv: 1907.02046 [cs.CL].

[72] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[73] Matthew E. Peters et al. *Deep contextualized word representations*. 2018. arXiv: 1802.05365 [cs.CL].

[74] Alec Radford. "Improving Language Understanding by Generative Pre-Training". In: 2018.

[75] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: (2018). URL: https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf.

[76] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2019. arXiv: 1910.10683 [cs.LG].

[77] Ines Montani Ramiro Gómez Sofie Van Landeghem. *spaCy 101*. 2019. URL: https://spacy.io/usage/spacy-101 (visited on 12/08/2019).

[78] Sebastian Ruder. *A Review of the Neural History of Natural Language Processing*. 2018. URL: https://towardsdatascience.com/lessons-learned-from-applying-deep-learning-for-nlp-without-big-data-d470db4f27bf (visited on 12/07/2019).

[79] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2016. URL: https://ruder.io/optimizing-gradient-descent (visited on 12/01/2019).

[80] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2016. arXiv: 1609.04747 [cs.LG].

[81] Sebastian Ruder. *On word embeddings - Part 1*. 2016. URL: https://ruder.io/word-embeddings-1/index.html#classicneurallanguagemodel (visited on 12/07/2019).

[82] Sebastian Ruder et al. "Transfer Learning in Natural Language Processing". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*. 2019, pp. 15–18.

[83] Arthur L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers". In: (1959).

[84] Constantin Hackober Sandra Faltl Michael Schimpke. *Universal Language Model Fine-Tuning (ULMFiT): State-of-the-Art in Text Analysis*. 2019. URL: https://humboldt-wi.github.io/blog/research/information_systems_1819/group4_ulmfit (visited on 12/16/2019).

[85] Tyler R. Scott, Karl Ridgeway, and Michael C. Mozer. *Adapted Deep Embeddings: A Synthesis of Methods for k-Shot Inductive Transfer Learning*. 2018. arXiv: 1805.08402 [cs.LG].

[86] Yashu Seth. *What makes the AWD-LSTM great?* 2018. URL: https://yashuseth.blog/2018/09/12/awd-lstm-explanation-understanding-language-model (visited on 12/16/2019).

[87] David Silver et al. “Mastering the game of Go without human knowledge”. In: *Nature* 550 (Oct. 2017), pp. 354–. URL: http://dx.doi.org/10.1038/nature24270.

[88] Leslie N. Smith. *Cyclical Learning Rates for Training Neural Networks*. 2015. arXiv: 1506.01186 [cs.CV].

[89] Department of State of United States of America Foreign Service Institute. *Language lerning difficulty for English speakers*. 2007. URL: http://web.archive.org/web/20071014005901/http://www.nvtc.gov/lotw/months/november/learningExpectations.html (visited on 11/23/2019).

[90] Ilya Sutskever. “Training Recurrent Neural Networks”. AAINS22066. PhD thesis. Toronto, Ont., Canada, Canada, 2013. ISBN: 978-0-499-22066-0.

[91] Jeremy Howard Terence Parr. *The Mechanics of Machine Learning*. https://mlbook.explained.ai. Terence Parr, 2019.

[92] Rachel Thomas. *A Code-First Introduction to Natural Language Processing*. 2019. URL: https://www.fast.ai/2019/07/08/fastai-nlp (visited on 12/23/2019).

[93] Paul Viola and Michael Jones. “Rapid object detection using a boosted cascade of simple features”. In: *Comput. Vis. Pattern Recog* 1 (Jan. 2001).

[94] Li Wan et al. “Regularization of Neural Networks Using Dropconnect”. In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML’13. Atlanta, GA, USA: JMLR.org, 2013, pp. III-1058–III-1066. URL: http://dl.acm.org/citation.cfm?id=3042817.3043055.

[95] Aleksander Wawer and Julita Sobiczewska. “Predicting Sentiment of Polish Language Short Texts”. In: Sept. 2019.

[96] Seth Weidman. *Deep Learning From Scratch: Building With Python First Principles*. O’Reilly, 2019.

[97] Eric Weisstein. *Convergent Sequence*. 2018. URL: http://mathworld.wolfram.com/ConvergentSequence.html (visited on 12/18/2019).

[98] Wikipedia contributors. *Anaphora (linguistics) — Wikipedia, The Free Encyclopedia*. [Online; accessed 23-December-2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=Anaphora_(linguistics)&oldid=931282202.

[99] Wikipedia contributors. *Bias–variance tradeoff — Wikipedia, The Free Encyclopedia*. [Online; accessed 15-December-2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=Bias%E2%80%93variance_tradeoff&oldid=930258148.

[100] Wikipedia contributors. *Document classification — Wikipedia, The Free Encyclopedia*. [Online; accessed 23-November-2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=Document_classification&oldid=916435721.

[101] Wikipedia contributors. *Feature learning — Wikipedia, The Free Encyclopedia*. [Online; accessed 24-November-2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=Feature_learning&oldid=923814694.

[102] Wikipedia contributors. *Gradient — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Gradient&oldid=929417044. [Online; accessed 5-December-2019]. 2019.

[103] Wikipedia contributors. *Hyperparameter (machine learning) — Wikipedia, The Free Encyclopedia*. [Online; accessed 2-December-2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=Hyperparameter_(machine_learning)&oldid=928631523.

[104] Wikipedia contributors. *Inflection — Wikipedia, The Free Encyclopedia*. [Online; accessed 29-December-2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=Inflection&oldid=927778150.

[105] Wikipedia contributors. *Naive Bayes classifier — Wikipedia, The Free Encyclopedia*. [Online; accessed 29-December-2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=Naive_Bayes_classifier&oldid=932675034.

[106] Wikipedia contributors. *Polysemy — Wikipedia, The Free Encyclopedia*. [Online; accessed 23-December-2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=Polysemy&oldid=930071048.

[107] Wikipedia contributors. *Principle of compositionality — Wikipedia, The Free Encyclopedia.* [Online; accessed 23-December-2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=Principle_of_compositionality&oldid=928640367.

[108] Wikipedia contributors. *Softmax function — Wikipedia, The Free Encyclopedia.* [Online; accessed 28-November-2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=Softmax_function&oldid=924981893.

[109] Wikipedia contributors. *Support-vector machine — Wikipedia, The Free Encyclopedia.* [Online; accessed 29-December-2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=Support-vector_machine&oldid=928737848.

[110] Katarzyna Wójcik and Janusz Tuchowski. "Comparison analysis of chosen approaches to sentiment analysis". In: *IT for practice* (2012), pp. 187–192.

[111] Xiao Yang, Craig Macdonald, and Iadh Ounis. "Using word embeddings in twitter election classification". In: *Information Retrieval Journal* 21.2-3 (2018), pp. 183–207.

[112] Jason Yosinski et al. *How transferable are features in deep neural networks?* 2014. arXiv: 1411.1792 [cs.LG].

[113] Kelly W. Zhang and Samuel R. Bowman. *Language Modeling Teaches You More Syntax than Translation Does: Lessons Learned Through Auxiliary Task Analysis.* 2018. arXiv: 1809.10040 [cs.CL].

Appendix A

Contents of CD

A compact disk attached to this work contains:

1. Source code of a document-level sentiment classification system for Polish text documents, `fincher`,
2. An included `Filmweb+` dataset stored in a `CSV` file, used for the purpose of training a domain-specific language model and a target classifier,
3. A documentation of the library methods used by the document-level sentiment classification system, available in `HTML` format,
4. A link to a repository containg the source code of a system used to generate the `Filmweb+` dataset (see appendix B).

Appendix B

Polish Film Reviews Crawler

A `Filmweb+` dataset used in this work was obtained by an use of implemented `Sorkin` system, which automatically downloads, pre-processes and stores Polish movie reviews fetched from major Polish blogs and websites, including:

- Filmweb,
- Film.org.pl,
- Kinoblog,
- FDB,
- BlogFilmowy24.

`Sorkin` system performs queries to a pre-defined set of websites using available library methods, then generates a list of new, previously unseen links to reviews based on the contents of collections in a document-based `MongoDB` database, and then downloads and stores new reviews in a text document format in either `rated` or `not_rated` collection, based on review details available.

The system's source code is available for download on GitHub.

Appendix C

Summary of the work in Polish

Celem niniejszej pracy dyplomowej było zaprojektowanie oraz zaimplementowanie systemu przeznaczonego do automatycznej klasyfikacji wydźwięku wypowiedzi, czyli ustalenia, czy dana wypowiedź jest pozytywna, negatywna bądź neutralna w odniesieniu do przedmiotu opinii, dla dokumentów tekstowych zawierających recenzje filmowe w języku polskim. Klasyfikacja opinii powinna zostać dokonana z dokładnością porównywalną z wiodącymi rozwiązaniami w swojej dziedzinie.

Zaimplementowany system został oparty o model klasyfikatora bazującego na głębokich, rekurencyjnych sieciach neuronowych. Jego trenowanie odbywa się w trójetapowym procesie, składającym się z:

1. Wstępnego trenowania *ogólnego modelu językowego* na ogólnym, wielodziedzinowym zbiorze danych *plwiki*, który został opracowany na bazie całej zawartości polskiej Wikipedii,
2. *Dostrajania* bazowego ogólnego modelu językowego przy użyciu dziedzinowego, autorskiego zbioru danych *Filmweb+*, w celu opracowania *dziedzinowego modelu językowego* dla recenzji filmowych,
3. Trenowania powstałego na bazie dziedzinowego modelu językowego *docelowego klasyfikatora* przy użyciu oetykietowanego podzbioru `Filmweb+`,

który wykorzystuje założenia uczenia nienadzorowanego i *transfer learningu* oraz metody regularyzacji rekurencyjnych sieci neuronowych opartych o komórki LSTM. Wykorzystywane korpusy danych zostały poddane procesowi *tokenizacji*, który uwzględnia fleksyjną naturę języka polskiego.

Zastosowane w pracy rozwiązania pozwalają na wytrenowanie docelowego klasyfikatora przy pomocy oetykietowanego zbioru danych, składającego się z zaledwie 3085 pozytywnych i 3085 negatywnych pod względem wydźwięku wypowiedzi przykładów recenzji filmowych, zawierających średnio 3512 znaków tekstu na recenzję (rozdział 5). Opracowany system osiągnął dokładność predykcji na zbiorze testowym wynoszącą 94,19%, stanowiąc tym samym najlepszy wynik w dziedzinie klasyfikacji wydźwięku wypowiedzi długich, zawierających średnio ponad 500 słów dokumentów tekstowych w języku polskim.

Podzielony na trzy rozdziały wstęp teoretyczny do pracy wprowadza czytelnika w zagadnienia dotyczące analizy wydźwięku wypowiedzi (rozdział 2), najważniejsze problemy współczesnego uczenia maszynowego (rozdział 3) oraz przetwarzania tekstu i analizy wydźwięku wypowiedzi opartego o rekurencyjne sieci neuronowe (rozdział 4).

W ramach pracy przeprowadzono także eksperymenty (rozdział 7), w których przebadano wpływ liczności słownika tokenów, ilości oetykietowanych danych w korpusie uczącym i różnych strategii uczenia oraz hiperparametrów na dokładność docelowego klasyfikatora. Zmierzono także czas obliczeń, niezbędnych do wykonania w celu wytrenowania docelowego klasyfikatora, który wyniósł mniej niż 10 godzin przy użyciu ogólnie dostępnych, konsumenckich kart graficznych wspierających technologię *CUDA*.

Finalnie w pracy przedstawiono dalsze, możliwe do obrania kierunki badań i eksperymentów, a także potencjalne komercyjne zastosowania systemu. Obecne rozwiązanie może zostać rozbudowane w system do analizy odbioru filmów i seriali przez polską widownię, bazując na recenzjach zamieszczonych na stronach internetowych. Z uwagi na swą modularną budowę, zaproponowany system może być zastosowany do klasyfikacji wydźwięku wypowiedzi w opiniach z innych dziedzin, takich jak np. polityka, czy analiza wpływu marki. W zależności od obranych korpusów uczących, opracowany system może być także zastosowany do dowolnie wybranego problemu klasyfikacji dokumentów tekstowych, jak np. klasyfikacja tematu czy wykrywanie niechcianych wiadomości e-mail w dowolnym języku. Wówczas, dla dziedziny analizy rynkowej, potencjalne aplikacje systemu obejmują jego użycie jako weryfikatora dla generatora propozycji popularnych tematów, produktów, czy artykułów, który określa, czy dana propozycja odpowiada na zapotrzebowanie rynkowe (rozdział 8).