National Technical University of Athens
School of Electrical and Computer Engineering
Division of Signals, Control and Robotics

Transfer Learning with Deep Neural Networks for Sentiment Analysis and Semantic Modeling

DIPLOMA THESIS

ALEXANDRA CHRONOPOULOU

Supervisor : Alexandros Potamianos
Associate Professor NTUA

Athens, March 2019

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Signals, Control and Robotics

Transfer Learning with Deep Neural Networks for Sentiment Analysis and Semantic Modeling

DIPLOMA THESIS

ALEXANDRA CHRONOPOULOU

Supervisor : Alexandros Potamianos
Associate Professor NTUA

Approved by the examining committee on the March 20, 2019.

Alexandros Potamianos Associate Professor NTUA	Ion Androutsopoulos Associate Professor AUEB	Georgios Stamou Associate Professor NTUA

Athens, March 2019

Alexandra Chronopoulou
Electrical and Computer Engineer

To my grandmother, Alexandra.

Abstract

In this work [1], we address the issue of poor performance in classification tasks due to scarcity of labeled data. To yield substantial improvements in classification tasks, we leverage pretrained representations and explore transfer learning methods, both in the form of pretrained classifiers and pretrained language models. We then present a more effective and refined transfer learning approach, where we introduce an auxiliary language model loss to the transferred model. The addressed issue is crucial in deep learning and has only recently been tackled in the Natural Language Processing field, as deep neural networks typically require an extended number of training annotated examples, yet large quantities of data are often expensive and difficult to collect.

First, we propose a method for successfully utilizing a pretrained sentiment analysis classification model to reduce the test error rate on an emotion recognition classification task. Transfer learning from pretrained classifiers exploits the representation that a model has learned for one supervised setting with plenty of data to obtain competitive results on a related task where only a small dataset is available. We aim to leverage the learned representation of the pretrained classifier to tackle the emotion classification task.

Next, we utilize pretrained representations from language models to address the same classification task. A learning algorithm can use information learned in the unsupervised phase to perform better in the supervised learning stage. Specifically, pretrained word representations captured by language models are useful as they encode contextual information and model syntax and semantics. We propose a three-step transfer learning method that includes pretraining a language model, fine-tuning it on the target task and transferring the model to a classifier to leverage these representations. We show an improvement of 10% on the WASSA 2018 emotion recognition dataset baseline. We achieve a F_1-score of 70.3%, ranking in the top-3 positions of the respective competition.

Finally, we present a conceptually simple and effective transfer learning approach that addresses the problem of catastrophic forgetting. Specifically, we combine the task-specific optimization functional with an auxiliary language model objective, which is adjusted during the training process. This preserves language regularities captured by language models, while enabling sufficient adaptation for solving the target task. The introduction of the auxiliary language model loss allows us to explicitly control the weighting of the pretrained part of the model and ensure that the distilled knowledge it encodes is preserved. Our approach shows robust results on 5 different classification datasets, where we report significant boosts over the baselines. The performance improvement is more pronounced when there is a mismatch between the pretraining and target task domains, which is the case in the *Sarcasm Corpus* dataset amongst others, where we achieve a F_1-score of 75%, 1% below state-of-the-art. We evaluate our model on a variety of classification tasks and demonstrate that our approach is able to yield impressive results even on a handful of training examples.

Key words

transfer learning, language modeling, emotion recognition, sentiment analysis, unsupervised pretraining, multi-task learning, deep learning, recurrent neural networks

[1] Papers: [1], [2], [3] have been conducted under the development of this thesis.

Acknowledgements

Throughout this thesis, I had the chance to be supervised by Prof. Alexandros Potamianos. I improved myself by listening to his advice and was able to publish my work. I want to thank him for his guidance and helpful comments. I would also like to thank PhD candidate Christos Baziotis for his patience and eagerness to answer my questions. Special thanks to my colleagues in NTUA-SLP lab for assisting me and for making our long hours of work, especially before deadlines, fun.

By presenting this thesis, I am officially finishing with an important part of my life and starting a new one. I have to thank and show my gratitude to my parents, for supporting me and helping me in all my decisions. I thank them in particular for instilling in me their love of knowledge and their principles.

Moreover, I want to thank Andreas for always being there, for sharing happy moments and difficulties with me, for believing in me and for motivating me to pursue my goals.

Finally, I would like to thank my friends. We have experienced amazing things together. As we were growing up together, we discussed a lot about the future and shared our ambitions. In retrospect, I feel lucky and grateful for these 6 (with some, even more) unforgettable years, and absolutely excited to see the rest.

Alexandra Chronopoulou,

Athens, March 20, 2019

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Transfer Learning

In the field of natural language processing, deep neural networks have improved the performance on a variety of tasks. However, learning a new model for each different task requires significant amounts of task-specific annotated data. To achieve top performance, these models typically need to be trained on more than millions of tokens annotated for each specific task, such as syntactic parsing [9], question answering [10, 11] and machine translation [12, 13].

However, in many real-world applications, there is a low availability of labeled data. In such scenarios, transfer learning offers an alternative solution, by leveraging the knowledge that a model has obtained solving one task to tackle a different, yet similar, task. Transfer learning has recently allowed a breakthrough in deep learning [14, 8], as it permits training networks in situations when limited training data are available. Transfer learning usually results in faster convergence and also achieves better performance than the model would have, if it was trained end-to-end on a small dataset. Moreover, it improves generalization, as pretrained models are purposefully trained on tasks that force the model to learn generic features that are useful in related contexts. Thus, when the model is transferred to a new task, it is less likely to overfit to the new training data. This generic pretrained feature representation is immensely important for natural language processing. Transfer learning surpasses the need of vast computational resources and extensive training.

A related direction is multi-task learning. Multi-task learning employs the similarities between different tasks (problems) to perform better in all of them. In deep learning, we usually aim to learn a highly informative feature representation of the input data and use it to make a prediction. By using multi-task learning, we might increase the performance of our model even further by forcing it to learn more generic representations. That way, implicit data augmentation is achieved, as the feature representation is learned not only by training on a specific dataset, but also on simultaneous training on datasets of similar tasks. In natural language processing, multi-task learning is used in a variety of applications, including machine translation [15, 16, 17], semantic parsing [18, 19, 20, 21], sequence labeling [22] and part-of-speech tagging [23].

1.2 Language Modeling

The goal of any transfer learning model is to obtain a generic feature representation while solving one task and use this knowledge to tackle a related task. A task which is suitable for pretraining is language modeling. Language modeling is a fundamental task of natural language processing, which provides a probability distribution over a sequence of words and offers a unique representation for each occurrence of a word, based on the context it is found in. It is thus able to encode the nuances of language, modeling syntax and semantics. Language modeling allows obtaining high-level feature representations and can provide a meaningful starting point for most supervised natural language processing tasks.

Moreover, language modeling receives every word of a sequence as input and essentially tries to predict the next word. Consequently, it does not require labeled training data, which are expensive and hard to collect. Raw text, in contrast, is abundantly available for every conceivable domain. Language

models can thus train on extensive corpora that are publicly available. Because of its ability to obtain generic word representations and the availability of unlabeled data, language modeling has shown to be effective for improving many natural language processing tasks [24, 25, 26, 27, 28]. Language modeling pretraining has notably improved results in natural language inference [29], named entity recognition [30], SQuAD question answering [31] and sentiment analysis [32].

1.3 Emotion Recognition & Sentiment Analysis

Emotion detection in natural language processing is the process of identifying discrete emotion expressed in text. Emotion analysis can be viewed as a natural evolution of sentiment analysis and its more fine-grained model. Sentiment analysis, with thousands of articles written about its methods and applications, is a well established field in natural language processing. It has proven very useful in several applications such as marketing, advertising [33, 34], question answering systems [35].

Understanding emotions is prevalent in every society. Examples of applications of emotion recognition can be found in political science [36], psychology, marketing [37], human-computer interaction [38] and many more. In marketing, emotion detection can be used to analyze consumers reactions to products and services to decide which aspect of the product should be changed to create a better relationship with customers in order to increase customer satisfaction [39]. Emotion detection, moreover, can be used in human-computer interaction and recommender systems to produce interactions or recommendations based on the emotional state of the user [40]. By understanding the important role of emotions in decision making process of humans, emotion detection can profit any entity or organization such as commercial institutes, political campaigns, managing the response to a natural disaster. It is also necessary to create better artificial intelligence tools, e.g. chatbots.

In the context of this thesis, we will be looking into transfer learning and multi-task learning techniques for natural language processing, proposing a novel method that combines them. Specifically, based on the fact that pretrained representations from unsupervised tasks are useful for supervised tasks with few training data, we propose a method to leverage pretrained feature representations to tackle different classification tasks. Specifically, we pretrain a language model on a generic Twitter corpus and transfer it to a variety of different classification tasks, in the fields of emotion recognition, sentiment analysis, emotion detection etc. We also introduce an auxiliary language model objective, to allow our model to learn more generic representations and avoid overfitting. By combining both a transfer learning and a multi-task learning approach, we propose a deep learning model able to tackle a variety of tasks and present competitive results.

1.4 Thesis Outline

The thesis is structured as follows. In Chapter 2 the machine learning background theory is presented. Specifically, the basic notions of machine learning are presented and then most traditional classification models are explained. After the traditional models are presented, deep learning is introduced, along with neural networks and especially recurrent neural networks (RNNs) and long short-term memory units (LSTMs). Finally, transfer learning and multi-task learning notions are presented, along with intuition about their usage. After the technical background is presented, Chapter 3 presents the natural language processing background needed to understand this thesis. After briefly presenting popular natural language processing tasks, language modeling is presented, initially in the form of a n-gram model based on the Markov assumption and then as a recurrent neural network. Then, transfer learning methods that are currently used to train natural language processing models are explained. As soon as transfer learning from language models is explained, we address in Chapter 4 an emotion recognition classification task and explain the methodology and transfer learning models implemented to tackle this task. After laying out the model architecture, experimental results and conclusions are presented. Afterwards, in Chapter 5, we propose a deep learning approach based on pretrained representations captured by language models that addresses the problem of catastrophic forgetting, by

adding an auxiliary language model objective to the transferred model. We leverage the abstract feature representations of language models and employ them to tackle downstream classification tasks. The results of our conducted experiments on five different classification datasets are then presented. Finally, in Chapter 6, our conclusions are presented and some ideas for future work are introduced.

Chapter 2

Machine Learning Background

2.1 Definition of Machine Learning

Machine learning (ML) is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine Learning algorithms allow computers to train on data inputs and use statistical analysis in order to output values that fall within a specific range. The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.

In ML, tasks are generally classified into broad categories. These categories are based on how learning is received or how feedback on the learning is given to the system developed. Two of the most widely adopted ML methods are *supervised learning* which trains algorithms based on example input and output data that is labeled by humans, and *unsupervised learning* which provides the algorithm with no labeled data in order to allow it to find structure within its input data.

2.2 Supervised Learning

In supervised learning, there are input variables (x) and an output variable (Y) and the goal is learning the mapping function from the input to the output via an algorithm.

$$Y = f(X) \tag{2.1}$$

The goal is to approximate the mapping function so well that when new input data (x) are fed to the model, the corresponding output variables (Y) could be successfully predicted. The process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers (entitled *labels*), the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

Supervised learning problems can be further grouped into *regression* and *classification* problems.

- Regression: the problem of predicting a continuous quantity. ($\boldsymbol{y}$ is continuous)
- Classification: deals with assigning observations into discrete categories, rather than predicting continuous quantities. In the simplest case, there are two possible categories; this case is known as binary classification. ($\boldsymbol{y}$ is categorical)

Supervised Learning with Neural Networks
In the case of neural networks, we restrict ourselves to search over specific families of functions, e.g. the space of all linear functions with d_{in} inputs and d_{out} outputs, which are called *hypothesis classes*. This restriction takes place because searching over the set of all possible functions is a very

hard (and rather ill-defined) problem. By restricting ourselves to a specific hypothesis class, we inject the learner with *inductive bias* - a set of assumptions about the form of the desired solution, as well as facilitating efficient procedures for searching for the solutions. One common hypothesis class is that of high-dimensional linear function, i.e. functions of the form:

$$f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{W} + \boldsymbol{b} \tag{2.2}$$
$$\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{d}_{in}}, \boldsymbol{W} \in \mathbb{R}^{\boldsymbol{d}_{in} \times \boldsymbol{d}_{out}}, \boldsymbol{b} \in \mathbb{R}^{\boldsymbol{d}_{out}}$$

where the vector $\boldsymbol{x}$ is the input of the function, while the matrix $\boldsymbol{W}$ and the vector $\boldsymbol{b}$ are the parameters. The goal of the learner is to set the values of the parameters $\boldsymbol{W}$ and $\boldsymbol{b}$ such that the function behaves as intended on a collection of input values $\boldsymbol{x}_{1:k} = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ and the corresponding desired outputs $\boldsymbol{y}_{1:k} = \boldsymbol{y}_1, \ldots, \boldsymbol{y}_k$. The task of searching over the space of functions is thus reduced to one of searching over the space of parameters. It is common to refer to parameters of the function as Θ. For the linear model case, $\Theta = \boldsymbol{W}, \boldsymbol{b}$.

2.3 Unsupervised Learning

In other machine learning problems, the training data consists of a set of input vectors $\boldsymbol{x}$ without any corresponding target values. So, the goal of unsupervised learning is to find patterns where the target values are unobservable, or infeasible to obtain. A large subclass of unsupervised tasks is the problem of clustering. *Clustering* refers to grouping observations together in such a way that members of a common group are similar to each other, and different from members of other groups. Another very interesting class of unsupervised tasks is generative modeling. *Generative models* are models that imitate the process that generates the training data. A good generative model would be able to generate new data that resemble the training data in some sense. This type of learning is unsupervised because the process that generates the data is not directly observable – only the data itself is observable.

2.4 Traditional Classification Models

2.4.1 Loss Function

The goal of any supervised learning algorithm is to return a function $f()$ that accurately maps input examples to their desired labels, i.e., a function $f()$ such that the predictions $\hat{\boldsymbol{y}} = f(x)$ over the training set are accurate. To make this more precise, the notion of a *loss function* is introduced, quantifying the loss suffered when predicting $\hat{\boldsymbol{y}}$ while the true label is $\boldsymbol{y}$. Formally, a loss function $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ assigns a numerical score (a scalar) to a predicted output $\hat{\boldsymbol{y}}$ given the true expected output $\boldsymbol{y}$. The loss function should be bounded from below, with the minimum attained only for cases where the prediction is correct. The parameters of the learned function are then set in order to minimize the loss L over the training examples (usually, it is the sum of the losses over the different training examples that is being minimized).

Given a labeled training set $(\boldsymbol{x_{1:n}}, \boldsymbol{y_{1:n}})$, a per-instance loss function L and a parameterized function $f(\boldsymbol{x}; \Theta)$, we define the corpus-wide loss with respect to the parameters Θ as the average loss over all training examples:

$$\mathcal{L}(\Theta) = -\frac{1}{N} \sum_{i=1}^{N} L(f(\boldsymbol{x_i}; \Theta), \boldsymbol{y_i}) \tag{2.3}$$

In this view, the training examples are fixed, and the values of the parameters determine the loss. The goal of the training algorithm is then to set the values of the parameters Θ, such that the value of $\mathcal{L}$ is minimized:

$$\hat{\Theta} = \arg\min_{\Theta} \mathcal{L}(\Theta) = \arg\min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} L(f(\boldsymbol{x_i}; \Theta), \boldsymbol{y_i}). \tag{2.4}$$

Now, we will define the notion of *entropy*. Suppose we want to communicate a set of n events from a particular probability distribution p of dataset X. Information entropy is the minimum average encoding size of information in dataset X to communicate the events [41]:

$$H(p) = \sum_x p(x) log \frac{1}{p(x)} \tag{2.5}$$

If entropy is high (encoding size is big on average), it means we have many inputs with small probabilities. It can be seen as measure of uncertainty, apart from amount of information.

Cross-entropy is the minimum average encoding size of communicating an event from one distribution for another distribution. Formally:

$$H_p(q) = \sum_x q(x) log \left(\frac{1}{p(x)} \right) \tag{2.6}$$

In the binary case (two categories) we define the *binary entropy*. It is the entropy of a Bernoulli process with probability p of two values. Let X be a random variable that can take only two values, 0 and 1. If $probability(X = 1) = p$, then $probability(X = 0) = 1 - p$ and the entropy is formally defined by:

$$\begin{aligned} H(X) &= p log \frac{1}{p} + (1-p) log \frac{1}{1-p} \\ &= -p log p - (1-p) log(1-p) \end{aligned} \tag{2.7}$$

Binary cross-entropy loss is used in binary classification with conditional probability outputs. We assume a set of two target classes labeled 0 and 1, with a correct label $y \in \{0, 1\}$. The classifier's output $\tilde{y}$ is transformed using the sigmoid (also called *logistic*) function $\sigma(x) = \frac{1}{1+e^{-x}}$ to the range [0, 1], and is interpreted as the conditional probability $\tilde{y} = \sigma(\tilde{y}) = P(y = 1|x)$. The prediction rule is:

$$\text{prediction} = \begin{cases} 0, & \text{if } \hat{y} < 0.5 \\ 1, & \text{if } \hat{y} \geq 0.5. \end{cases}$$

The network is trained to maximize the log conditional probability $\log P(y = 1|\boldsymbol{x})$ for each training example ($\boldsymbol{x}$, y). The logistic loss is defined as:

$$L_{\text{logistic}}(\hat{y}, y) = -y \log \hat{y} - (1-y) \log(1-\hat{y}). \tag{2.8}$$

When we use logistic loss, we assume that the output layer is transformed using the sigmoid function.

The *categorical cross-entropy loss* (also referred to as *negative log likelihood*) is used when a probabilistic interpretation of the scores is desired. Let $\boldsymbol{y} = \boldsymbol{y}_{[\mathbf{1}]}, \ldots, \boldsymbol{y}_{[\boldsymbol{n}]}$ be a vector representing the true multinomial distribution over the labels $\mathbf{1}, \cdots, \boldsymbol{n}$, and let $\hat{\boldsymbol{y}} = \hat{\boldsymbol{y}}_{[\mathbf{1}]}, \ldots, \hat{\boldsymbol{y}}_{[\boldsymbol{n}]}$ be the linear classifier's output, which was transformed by the softmax function and represent the class membership conditional distribution $\hat{\boldsymbol{y}}_{[\boldsymbol{i}]} = P(y = 1|\boldsymbol{x})$. Then, the categorical cross-entropy loss for the n^{th} sample is:

$$L_{\text{cross-entropy}}(\hat{\boldsymbol{y}}_{\boldsymbol{i}}, \boldsymbol{y}_{\boldsymbol{i}}) = -\boldsymbol{y}_{[\boldsymbol{i}]} \log(\hat{\boldsymbol{y}}_{[\boldsymbol{i}]}) \tag{2.9}$$

To optimize the parameters of our model, we seek to maximize its likelihood, or else minimize the average of the negative log likelihood of all the available N training samples. The objective function takes the following form:

$$L_{\text{cross-entropy}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\frac{1}{N} \sum_{i=1}^{N} \boldsymbol{y}_{[\boldsymbol{i}]} \log(\hat{\boldsymbol{y}}_{[\boldsymbol{i}]}) \tag{2.10}$$

where N is the number of training samples.

When training Neural Networks (NN), Equation 2.10 is particularly useful. The value of the loss function $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ allows computing the error of the NN in terms of the classification decisions made for the N samples. Instead of using the whole dataset in every iteration of the training process, we usually compute the error over subsets off the training set (called *batches*). To learn the optimal parameters for our models, we employ a parameter optimizer based on the computation of the gradient $\nabla_\theta L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ to find a local minimum. The most popular algorithm in order to optimize the weights is *backpropagation* [42]. The backpropagation algorithm relies on methodically computing the partial derivatives (gradients) of each layer of a NN with respect to the parameters that need to be tuned using the chain rule, in order to minimize the loss. Their weights are then updated accordingly. The error computed by the partial derivatives configures the amount of alteration of the weights.

What we are actually trying to do when employing the backpropagation algorithm is to approximate the local minima of the non-linear minimization problem. This problem cannot be solved in polynomial time by any algorithm (it belongs to a group of problems called NP-problems).

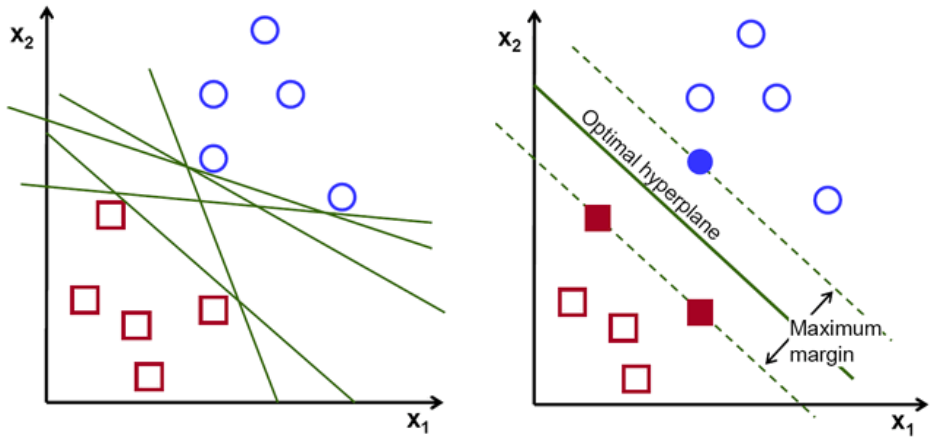
2.4.2 Support Vector Machines (SVMs)

Suppose we have a two-class classification problem using linear models of the form

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\boldsymbol{T}} \cdot \boldsymbol{\phi}(\boldsymbol{x}) + b \tag{2.11}$$

where $\boldsymbol{\phi}(\boldsymbol{x})$ denotes a fixed feature-space transformation, and we have made the bias parameter b explicit. The training dataset comprises N input vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ with corresponding target values $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$ where $y_i \in \{-1, 1\}$, and new data points $\boldsymbol{x}$ are classified according to the sign of $f(\boldsymbol{x})$. We shall assume for the moment that the training data set is linearly separable in feature space, so that by definition there exists at least one choice of the parameters w and b such that a function of the form 2.11 satisfies $f(\boldsymbol{x_i}) > 0$ for points having $y_i = +1$ and $f(\boldsymbol{x_i}) < 0$ for points having $y_i = -1$, so that $y_i f(\boldsymbol{x_i}) > 0$ for all training points.

An example of a training set is shown in Figure 2.1a, where the samples that belong to the first class are red and square, while the samples that belong to the second class are blue and circular.



(a) Possible separation hyperplanes. **(b)** A maximal margin linear classifier.

Figure 2.1: Example of binary classification of two linearly separable classes.

There are infinite possible straight lines that separate the two classes. The goal of the SVM algorithm is to find the most generalized separator. In other words, the SVM algorithm tries to find the hyperplane for which the minimum distance between the two classes (*margin*) is as wide as possible. The hyperplane that satisfies the above condition is the optimal hyperplane and can be shown in Figure 2.1b.

If $f(\boldsymbol{x})$ separates the data, the geometric distance between a point $\boldsymbol{x_i}$ and a hyperplane $f(\boldsymbol{x}) = 0$ is $\frac{|f(\boldsymbol{x_i})|}{||\boldsymbol{w}||}$. Furthermore, we are only interested in solutions for which all data points are correctly classified, so that $y_i f(\boldsymbol{x_i}) > 0$ for all i. Then, the distance between a point $\boldsymbol{x_i}$ and the optimal hyperplane is given by:

$$\frac{y_i f(\boldsymbol{x}_i)}{||\boldsymbol{w}||} = \frac{y_i(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_i) + b)}{||\boldsymbol{w}||} \tag{2.12}$$

The margin is given by the perpendicular distance to the closest point xn from the dataset, and we wish to optimize the parameters $\boldsymbol{w}$ and b in order to maximize this distance. Thus the maximum margin solution is found by solving

$$L(\boldsymbol{w}, \mathbf{b}) = \arg\max_{\boldsymbol{w},\mathbf{b}} \left\{ \frac{1}{||\boldsymbol{w}||} \min_i \left\{ y_i(\boldsymbol{w}^{\mathrm{T}} \phi(\boldsymbol{x}_i) + \mathbf{b}) \right\} \right\} \tag{2.13}$$

Maximizing $\frac{1}{||\boldsymbol{w}||}$ is equivalent to minimizing $\frac{1}{2}||\boldsymbol{w}||^2$. Our problem now becomes:

$$L(\boldsymbol{w}) = \arg\min_{\boldsymbol{w},\mathbf{b}} \left\{ \frac{1}{2}||\boldsymbol{w}||^2 \right\} \tag{2.14a}$$

$$y_i(\boldsymbol{w}^{\mathrm{T}} \phi(\boldsymbol{x}_i) + \mathbf{b}) \geq 1, \qquad i = 1, \ldots, N. \tag{2.14b}$$

The solution of equation 2.14a is given by Lagrange multipliers [43].

2.4.3 Logistic Regression (LR)

When it comes to classification, we are determining the probability of an observation to be part of a certain class or not. Therefore, we wish to express the probability with a value between 0 and 1. A simple classification algorithm that generates values of that form is Logistic Regression (LR) classifier.

Suppose we have a simple problem for a binary classifier, as the one described earlier in the same Section. Let $\boldsymbol{x}_{i=1:N} = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ be the input vectors where $\boldsymbol{y}_i \in \{0, 1\}$. The activation of the LR classifier is determined by applying a sigmoid function over the fitted line in order to get the final classification decision. As described in **SVMs**:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.15}$$

The activation of LR for a given vector $\boldsymbol{x}$ is defined as follows:

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T \boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{w}^T \boldsymbol{x}}} \tag{2.16}$$

The loss function to be minimized while fitting the LR to the training data is the following:

$$\frac{1}{2}||\boldsymbol{w}|| + C \sum_{i=1}^{N} log(exp(-y_i(\boldsymbol{w}^T \boldsymbol{x}_i + \mathbf{b})) + 1) \tag{2.17}$$

where C > 0 and b represent the coefficients of penalization of wrong classifications and the intercept of the hyperplane.

LR is a highly efficient technique, that does not require vast computational resources, tuning, or regularization. It thus provides a solid baseline for most NLP tasks. An obvious disadvantage is that LR cannot solve non-linear problems, since its decision surface is linear. A common "trick" used in order to apply LR to multi-class classification is to iteratively treat each pair of classes as a binary classification problem, to which LR is performed.

2.5 Deep Neural Networks

2.5.1 Introduction

Deep learning is a set of learning methods attempting to model data with complex architectures combining different non-linear transformations. The elementary bricks of deep learning are the neural networks, that are combined to form the deep neural networks. These techniques have enabled significant progress in the fields of sound and image processing, including facial recognition, speech recognition, computer vision, automated language processing, text classification. Potential applications are numerous. A spectacular example is the AlphaGo program, which learned to play the go game by the deep learning method, and beat the world champion in 2016.

2.5.2 Artificial Neural Networks

An Artificial Neural Network (ANN) is a biologically inspired computational model that is patterned after the network of neurons present in the human brain. The area of ANNs has originally been inspired by the goal of modeling biological neural systems, but has since diverged and become a matter of engineering and achieving good results in Machine Learning tasks. Thus, we first introduce a very brief and high-level description of the biological system that have influenced a large portion of Machine Learning.

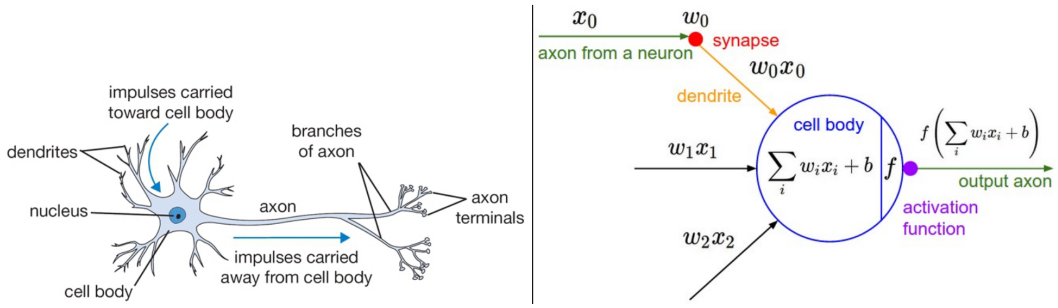


Figure 2.2: A biological neuron (left) and its mathematical notation (right). Figure from [4].

The basic computational unit of the brain is a *neuron*. Billions of neurons can be found in the human nervous system. Figure 2.2 shows the comparison between a biological neuron and its mathematical notation. Each neuron receives input signals from its *dendrites* and produces output signals along its (single) *axon*. The axon connects via synapses to dendrites of other neurons. In the computational model of a neuron, the signals that travel along the axons (e.g. x_0) interact multiplicatively (e.g. w_0x_0) with the dendrites of the other neuron based on the synaptic strength at that synapse (e.g. w_0). The idea is that the synaptic strengths (the weights w) are learnable and control the strength of influence of one neuron on another. In the basic model, the dendrites carry the signal to the cell body where they all get summed. If the final sum is above a certain threshold, the neuron can fire, sending a spike along its axon. In the computational model, we assume that only the frequency of the firing communicates information. We thus model the firing rate of the neuron with an *activation function* f, which represents the frequency of the spikes along the axon. A common choice of activation function is the sigmoid function σ, since it takes a real-valued inputs and squashes it to range between 0 and 1.

In order to learn complex non-linear functions, architectures that combine several artificial neurons can be designed and implemented. Such architectures are called Multi-Layer Perceptrons (MLPs). Instead of MLPs, Feed-Forward Neural Networks (FFNNs) can also be implemented, where each neuron connects with all neurons of the previous layer and there are no connections between the neurons of the same layer.

Each neural network is composed of the following layers, as depicted in Figure 2.3:

- *Input layer*: This layer accepts the input data. It provides information from the outside world to

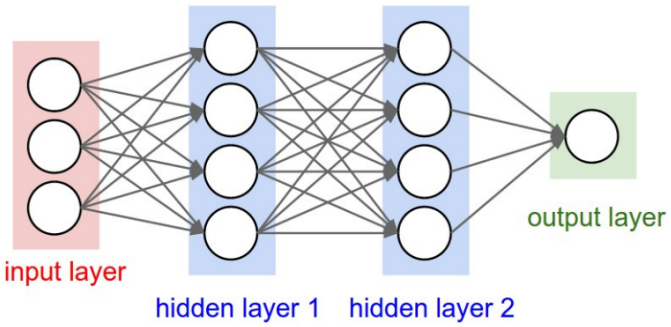


Figure 2.3: A three-layer Neural Network. It contains three inputs, two hidden layers of four neurons each and one output layer. Figure from [4].

the network without any further computation. Nodes just pass on the information to the hidden layer.

- *Hidden layer(s)*: There can be more than one hidden layers, which are used to preprocess the inputs obtained by the previous layer. They extract the required features from the input data. When moving to higher hidden layers, higher-level features are constructed.
- *Output layer*: After the data is preprocessed, a decision is made by the network in this layer.

Activation function

Activation functions are nodes that decide whether a neuron should be activated or not by introducing non-linearities to its output. A neural network needs to add non-linear functions in order to perform complex tasks and give accurate results. The functions that are commonly used as activation functions are the following:

- *Sigmoid.*

  The sigmoid non-linearity has the mathematical form:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.18}$$

  it takes a real-valued number and squashes it into range between 0 and 1. The function is graphically illustrated in Figure 2.4.

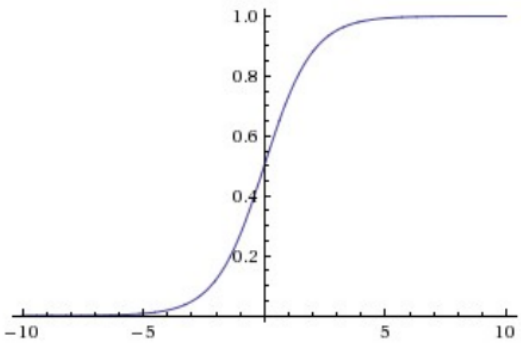


Figure 2.4: The sigmoid function.

Sigmoid function, however, has two major drawbacks: 1) at either tail of 0 or 1, the neuron's activation saturates and the gradient is almost zero. In other words, the gradient "vanishes" and training stops. 2) The output of sigmoid is not centered to zero. Therefore, since the data coming into a neuron is always positive, the gradient of the weights will either all be positive, or all negative and undesirable zig-zagging dynamics would be introduced to the network.

- *Hyperbolic tangent (tanh).*

  The tanh non-linearity is shown on Figure 2.5. It squashes a real-valued number to the range [-1, 1]. The advantage of this function is that the values of tanh are zero-centered, which helps the next neuron during propagating. The tanh activation function follows this function:

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.19}$$

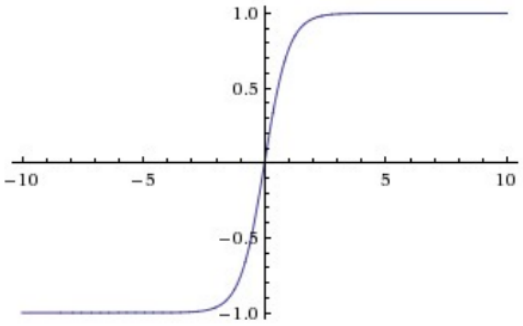


Figure 2.5: The tanh function.

Tanh function is simply a scaled sigmoid function, where $tanh(x) = 2\sigma(2x) - 1$. Tanh function is obviously zero-centered. Still, the saturation problem persists.

- *Rectified Linear Unit (ReLU).*

  ReLU is one of the most widely used activation functions. It computes the function:

$$f(x) = x^+ = \max(0, x) \tag{2.20}$$

  In other words, the activation is simply thresholded at zero.

  ReLU does not involve computationally expensive operations (exponentials etc.) and thus converges faster. It is also sparsely activated, which is desirable because for any negative input, the function will not activate at all, thus making it more likely that neurons are actually processing meaningful aspects of the problem (or are not "fired" at all). It also avoids the saturation problem, due to its linear form. However, ReLU function is can easily cause the neurons to be "stuck". If the gradients computed are negative, then the output is always zero and the neuron is unlikely to recover.

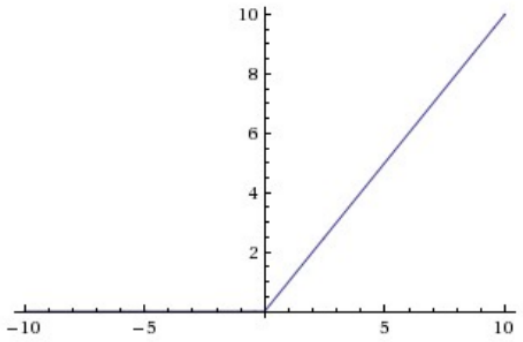


Figure 2.6: The ReLU function.

- *Leaky ReLU.*

  Instead of the function being zero when x < 0, a leaky ReLU will instead have a small negative slope (of $a = 0.01$, or so). That is, the function computes:

$$\text{f(x)} = \begin{cases} x, & \text{if } x > 0 \\ ax, & \text{otherwise} \end{cases}$$

  So, Leaky ReLU fixes the "dying ReLU" problem, as it does not have zero-slope parts.

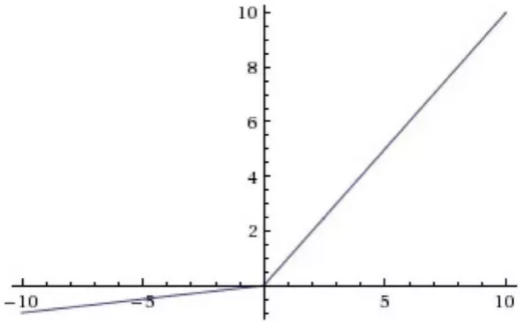


Figure 2.7: The Leaky ReLU function.

Regularization

Equation 2.4 tries to minimize the loss and may result in *overfitting* the training data. Overfitting occurs when we have such a rich model family that we do not just fit the underlying function, but we in fact fit the noise as well. It naturally results in a model that does not generalize well on test data.

Regularization is a way to mitigate this undesirable behavior. To face potential loss of generalization, we impose restrictions to the form of the solution. Specifically, the loss function takes the following form:

$$\hat{\Theta} = \arg\min_{\Theta} \mathcal{L}(\Theta) = \arg\min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} L(f(\boldsymbol{x_i}; \Theta), \boldsymbol{y_i}) + \lambda R(\Theta). \tag{2.21}$$

The regularization term considers the parameter values and scores their complexity. We then look for values that have both a low loss and low complexity. What regularization inherently intends to do is penalize complex models and favor simpler ones.

λ is a value that has to be set manually, based on the classification performance on a development set (called *hyperparameter*). The regularizers *R* measure the norms of the parameter matrices and opt for solutions with low norms. The 2 most common regularization norms are:

- L_2 *regularization*:

$$R_{L_2}(\boldsymbol{W}) = ||\boldsymbol{W}||_2^2 = \sum_{i,j} (\boldsymbol{W}_{[i,j]})^2. \tag{2.22}$$

  R takes the form of the standard Euclidean norm (L_2-norm) of the parameters, trying to keep the sum of the squares of the parameter values low. Large model weights $\boldsymbol{W}_{[i,j]}$ will be penalized, since they are considered "unlikely". L_2 is often referred to as *weight decay*. As one can observe from Equation 2.22, high weights are severely penalized, but weights with small values are only negligibly affected.

- L_1 *regularization*:

$$R_{L_1}(\boldsymbol{W}) = ||\boldsymbol{W}||_1 = \sum_{i,j} |\boldsymbol{W}_{[i,j]}|. \tag{2.23}$$

  The L_1 regularizer punishes uniformly low and high values and intends to decrease all non-zero parameter values towards zero. So, it encourages sparse solutions, or else models with many parameters with a zero value. The L_1 regularizer is also called a *sparse prior* or *lasso* [44].

- *Dropout:*

  An effective technique for preventing neural networks from overfitting the training samples is *dropout* training [45, 46]. Dropout is designed to prevent the network from learning to rely on specific weights. It randomly sets to zero (drops) half of the neurons in the network (or in a specific layer) in each training example, in the stochastic-gradient training. The dropout technique is one of the key factors contributing to very strong results of neural networks.

Optimization

In order to train the model, we need to solve the optimization problem in Equation 2.21. A common solution is to use a gradient-based method. A *gradient* (at a point) is the slope of the tangent to the function at that point. It points to the direction of largest increase of the function. Gradient-based methods seek to minimize the objective function $\mathcal{L}(\Theta)$ by repeatedly computing an estimate of the loss $\mathcal{L}$ over the training set, computing the gradients of the parameters Θ of the model with respect to the loss estimate and updating the parameters in the opposite direction of the gradient.

Gradient descent (GD) is one of the most popular algorithms to perform optimization in neural networks. It computes the gradient of the cost function with respect to the parameters θ for the entire dataset. The learning rate (η) is a hyperparameter that controls the extent to which the parameters of the model are adjusted with respect to the loss gradient. GD is formally defined as:

$$\theta = \theta - \eta \nabla_\theta J(\theta) \tag{2.24}$$

Stochastic Gradient Descent (SGD) [47] in contrast performs a parameter update for each training example $\boldsymbol{x}_i$ and label $\boldsymbol{y}_i$:

$$\theta = \theta - \eta \nabla_\theta J(\theta; x_i; y_i) \tag{2.25}$$

Gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. SGD does away with this redundancy by performing one update at a time. It is therefore usually much faster and can also be used to learn online.

Vanilla gradient descent, however, does not guarantee good convergence. The learning rate needs to be carefully tuned, as a small value leads to slow convergence, while a very large value can hinder convergence and cause the loss function to fluctuate around the minimum or even diverge. Moreover, the same learning rate applies to all parameter updates. If our data is sparse and our features have very different frequencies, it is possible that we do not want to update them to the same extent, but we instead want to perform larger updates for rarely occurring features. Finally, a key challenge of minimizing highly non-convex error functions common for neural networks is avoiding getting trapped in local minima that are suboptimal.

A number of alternate optimization algorithms have been introduced to tackle these problems. Currently, optimization techniques with automatic regulation of the learning rate are used such as Adagrad [48], Adadelta [49] and Adam [50], which is the most widely used optimization method in Neural Networks for NLP.

Backpropagation

To minimize the cost function $J(\theta)$ of a given neural network using the optimal set of values for θ (weights, also denoted as w), we need to compute the gradient. Even though the mathematics of

gradient computation for neural networks simply follow the chain-rule of differentiation, they can become laborious and error-prone for complex networks. Fortunately, gradients can be efficiently computed using the *backpropagation* algorithm [51, 52]. Backpropagation methodically computes the derivatives of a complex expression using the chain-rule, while caching intermediary results. A neural network is essentially a mathematical expression and can therefore be represented as a computation graph, where each node corresponds to a specific weight of the network. To update the weights of the network after each computation of the loss function, the key element is an expression for the partial derivative $\frac{\partial J}{\partial w}$ of the loss function J with respect to any weight w in the network. By inspecting the values of the partial derivatives, we gain intuition towards the sensitivity of the loss function with respect to those parameters. The gradients are a measure of how well the neural network is performing and help us in fine tuning the weights of the network to minimize the loss and find a model that fits our data.

2.5.3 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are a powerful and robust type of neural networks, that are particularly useful because of their internal memory. Connections between units of a RNN form a directed graph along a sequence. This allows it to exhibit temporal dynamic behavior for a time sequence. RNNs use their internal state (memory) to process sequential data as inputs. Intuitively, RNNs are able to remember important things about the input they received which enables them to make precise predictions for the data that comes next. As one can observe in Figure 2.9, basic RNNs are nodes

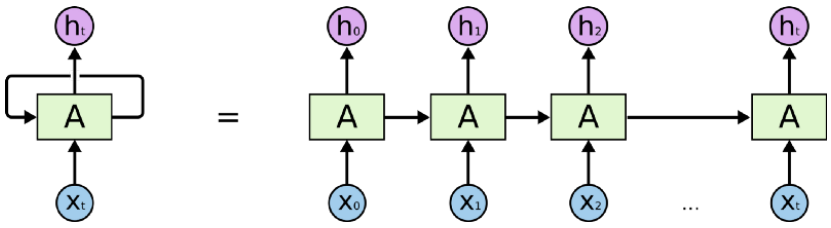


Figure 2.8: Unfolded basic recurrent neural network. Source: http://colah.github.io

organized into successive "layers". The RNN first takes the x_0 from the sequence of input and it outputs h_0 (hidden state). The hidden state h_0 along with next input x_1 is the input for the next step. Accordingly, h_1 along with x_2 is the input for the next step and so on. So, the RNN remembers the context of the input it has already seen while training.

Formally, at each time step t, the equations that describe the function of the RNN are:

$$h_t = f_h(W_{hh}h_{t-1} + W_{hx}x_t + b_h) \tag{2.26}$$

$$y_t = f_y(W_{yh}h_t + b_y) \tag{2.27}$$

where h_t is the hidden state at time step t, x_t is the input vector at time step t, y_t is the output vector at time step t, b_h is the bias for h, b_y is the bias for y and f_x, f_h are the activation functions for x and h respectively. The are three separate matrices of weights: W_{hx} (input-to-hidden weights), W_{hh} (hidden-to-hidden), and W_{yh} (hidden-to-output).

Bi-directional RNNs

As mentioned above, RNNs capture information about the sequential data they have seen until time step t and encode it in their hidden state. However, it is also possible to acquire more information by reading a given sequence backwards, in order to make more accurate predictions. So, a bi-directional RNN is obtained as follows:

We encode the input sequence from the beginning to the end (forward RNN) and also in reverse (backward RNN). We then combine the hidden states of the two RNN layers in order to find the hidden state for each time step. Specifically, we separately compute the hidden state of the forward RNN $\overrightarrow{h_t}$

at time step t as well as the corresponding hidden state of the backward RNN $\overleftarrow{h_t}$ and concatenate them in order to compute the final hidden state at each timestep. To this end, the hidden state at time step t is simply the concatenation of the two vectors, namely: $h_t = \overrightarrow{h_t} || \overleftarrow{h}_{T-t}$. The same applies for all the $T+1$ time steps of the input sequence.

Long Short-Term Memory (LSTM) unit

In theory, classic (or "vanilla") RNNs can keep track of arbitrary long-term dependencies in the input sequences. The problem of vanilla RNNs is computational (or practical) in nature: when training a vanilla RNN using backpropagation, the gradients which are back-propagated can "vanish" (that is, they can tend to zero) or "explode" (that is, they can tend to infinity), because of the computations involved in the process, which use finite-precision numbers.

LSTM networks introduced by Hochreiter and Schmidhuber [53] partially overcome these problems by preserving long-distance dependencies between words and distilling words that are not important from the cell gate through its forget gate layer.

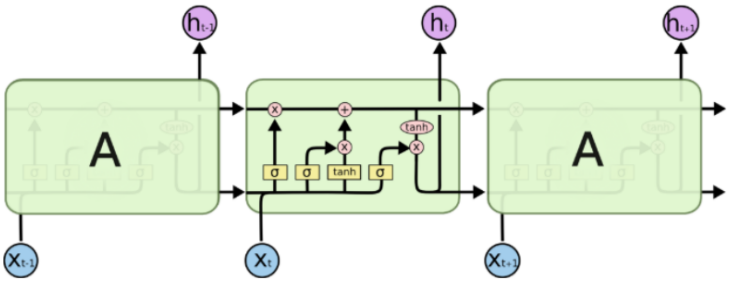


Figure 2.9: The repeating module in an LSTM.

Given a sequence $x_1, x_2, ..., x_t, ..., x_n$ of vectors of an input sequence of length n, for vector x_t, with inputs h_{t-1} and c_{t-1}, h_t and c_t are computed as follows:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{2.28}$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{2.29}$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{2.30}$$

$$u_t = tanh(W_u x_t + U_u h_{t-1} + b_u) \tag{2.31}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot u_t \tag{2.32}$$
$$h_t = o_t \odot tanh(c_t) \tag{2.33}$$

- **Forget gate** (f_t): This gate decides what information should be thrown away or kept. Information from the previous hidden state h_{t-1} and information from the current input x_t is passed through the sigmoid activation function. Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.
- **Input gate** (i_t): The previous hidden state and current input are passed into a sigmoid function, that decides which values will be updated by forcing the values to be between 0 and 1 (0 means unimportant and 1 means important). The hidden state and current input are also passed to the tanh function to squish values between -1 and 1 (u_t). Finally, the tanh output is multiplied with the sigmoid output ($i_t \odot u_t$). The sigmoid output will filter the important information of tanh.
- **Cell state** (c_t):The cell state gets pointwise multiplied by the forget vector. This has a possibility of dropping values in the cell state if it gets multiplied by values near 0. Then we take the output from the input gate and do a pointwise addition which updates the cell state to new values that the neural network finds relevant.

- **Output gate** (o_t): The output gate decides what the next hidden state should be. As the hidden state contains information on previous inputs, it is also used for predictions. First, the previous hidden state and the current input are passed into a sigmoid function. Then, the newly modified cell state is passed to the tanh function. We multiply the tanh output with the sigmoid output ($o_t \odot tanh(c_t)$) to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.

Self-Attention mechanism

The basic idea behind attention is that not all vectors in a given sequence contribute to the same degree to the meaning that is expressed in the overall input. So, the model should not use all vectors equally to make a prediction, but focus on the parts of the input that contain the most relevant information for a given task. To implement this approach, an attention mechanism [54, 13] can be used in order to find the relative importance of each input vector of a sequence. To amplify the contribution of the most informative vectors, we assign a weight a_i to the hidden step that corresponds to each vector h_i. We compute the fixed representation r of the whole input sequence, as the weighted sum of all hidden states.

$$e_i = tanh(W_h h_i + b_h), \quad e_i \in [-1, 1] \tag{2.34}$$

$$a_i = \frac{exp(e_i)}{\sum_{t=1}^{T} exp(e_t)}, \quad \sum_{i=1}^{T} a_i = 1 \tag{2.35}$$

$$r = \sum_{i=1}^{T} a_i h_i \tag{2.36}$$

where W_h and b_h are the attention layer's weights.

2.6 Transfer Learning

Motivation

In Machine Learning (and in particular Deep Learning), a ubiquitous problem is that models which solve complex problems need vast amounts of data. Nevertheless, getting these amounts of data for supervised models can become unfeasible due to time restrictions or computational limitations. Moreover, models that are trained on small, specific datasets face a performance drop, when they are used to tackle a different task, which might still be similar to the one they were trained on.

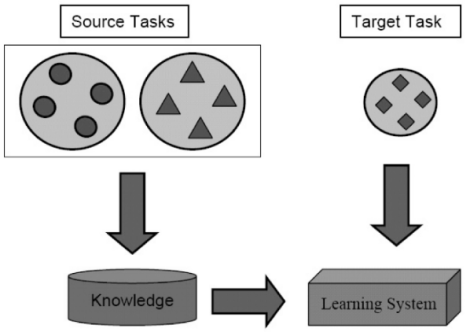


Figure 2.10: Transfer learning technique.

The goal of transfer learning is to improve learning in the *target* task by leveraging knowledge from the *source* task, as depicted in Figure 2.11.

Definition

In transfer learning, the concepts of a domain and a task are used. A domain D consists of a feature space X and a marginal probability distribution $P(X)$ over the feature space, where $X = x_1, \ldots, x_n \in X$. Given a domain $D = \{X, P(X)\}$, a task T consists of a label space Y and a conditional probability distribution $P(Y|X)$ that is typically learned from the training data $\{x_i, y_i\}$, with $x_i \in X$ and $y_i \in Y$.

Given a source domain D_S, a corresponding source task T_S, as well as a target domain D_T and a target task T_T, transfer learning's objective is to enable us to learn the target conditional probability distribution $P(Y_T|X_T)$ in D_T with the information gained from D_S and T_S, where $D_S \neq D_T$ or $T_S \neq T_T$.

Qualitative analysis

There are three common measures by which transfer might improve learning, that are depicted in Figure 2.11. First is the initial performance achievable in the target task using only the transferred knowledge, before any further learning is done, compared to the initial performance of an ignorant agent. Second is the amount of time it takes to fully learn the target task given the transferred knowledge compared to the amount of time to learn it from scratch. Third is the final performance level achievable in the target task compared to the final level without transfer. [5]

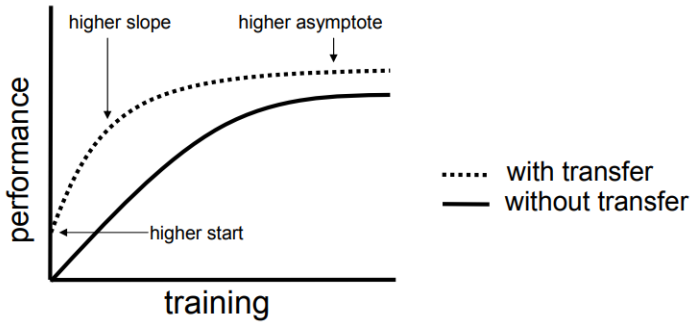


Figure 2.11: Three ways in which transfer might improve learning, according to [5].

The case of unsupervised pretraining

A special case of transfer learning involves a scenario where source task is unsupervised and target task is supervised. This scenario is particularly interesting because we often have very large amounts of unlabeled training data, yet relatively little labeled training data. Training with supervised techniques on the labeled subset often results in severe overfitting. By learning good representations from the unlabeled data, we can perform better in the supervised learning task [55].

This transfer learning case is called *unsupervised pretraining*. This procedure is an example of how a representation learned for one task (unsupervised learning, trying to capture the shape of the input distribution) can sometimes be useful for another task (supervised learning). It is called *pretraining*, because it is supposed to be only a first step before a joint training algorithm is applied to *fine-tune* all the layers together. In the context of supervised learning task, it can be viewed as a regularizer and a form of parameter initialization.

- **Regularizing effect**: It is possible that pretraining initializes the deep neural network in a location that would otherwise be inaccessible - for example, a region that is surrounded by areas where the cost function varies so much from one example to another that only a very noisy estimate of the gradient is given.

- **Parameter initialization**: Pretraining, in most cases, improves performance on the supervised task. The basic idea is that some features that are useful for the unsupervised task may also be useful for the supervised learning task.

2.7 Multi-task Learning

In multi-task learning (MTL)[56], we have several related predictions tasks. We would like to leverage the information in one of the tasks in order to improve the accuracy on the other tasks. In deep learning, the idea is to have different networks for the different tasks, but let the networks share some of their structure and parameters. This way, a common predictive core (the shared structure) is influenced by all the tasks, and training data for one task may help improve the predictions of the other ones.

In the context of deep learning with neural networks, multi-task learning is typically done with either *hard* or *soft* parameter sharing.

Hard parameter sharing
It is generally applied by sharing the hidden layers between all tasks, while keeping several task-specific output layers.

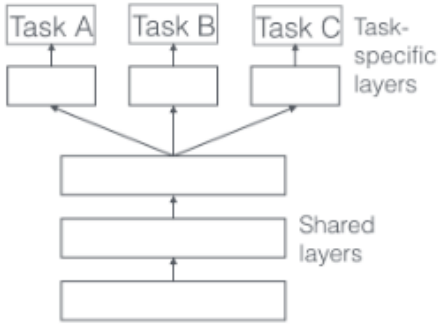


Figure 2.12: Hard parameter sharing for MTL in deep neural networks. [6].

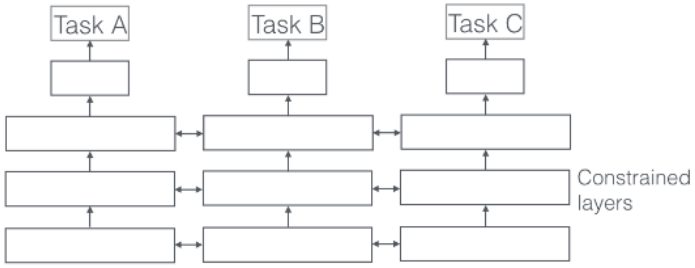


Figure 2.13: Soft parameter sharing for MTL in deep neural networks. [6].

Hard parameter sharing reduces the risk of overfitting. The intuition behind this is that the more tasks we are learning simultaneously, the more our model has to find a common representation that captures all of the tasks and the less our chance of overfitting on our original task.

Soft parameter sharing
On the other hand, in soft parameter sharing, each task has its own model and parameters. The distance between the parameters of the model is regularized in order to encourage the parameters to be similar [57].

Chapter 3

Natural Language Processing Background

3.1 Definition of Natural Language Processing

Natural Language Processing (NLP) is a field of computer science, artificial intelligence (also called machine learning), and linguistics concerned with the interactions between computers and human (natural) languages. It is the process of a computer extracting meaningful information from natural language input and/or producing natural language output. It is analysis of human language based on semantics and various parsing techniques [58]. The goal of NLP is to identify the computational machinery needed for an agent to exhibit various forms of linguistic behavior. It also design, implement, and test systems that process natural languages for practical applications. NLP is a discipline between linguistics and computer science which is concerned with the computational aspects of the human language faculty. The main task of it is to construct programs in order to process words and texts in natural language. The main aspects of NLP are:

- Information Retrieval (IR): It is concerned with storing, searching and retrieving of information from text documents. It is a field within computer science closer to databases and relies on some of the NLP methods.
- Machine Translation (MT): It is related to automatic translation from one human language to another [59].
- Language Analysis: It is concerned with parsing of an input sentence to construct syntactic tree. and further sentiment analysis is done to find meaningful words in a sentence.

The levels of language that NLP examines are:

- Phonology: This level deals with the interpretation of speech sounds within and across words. There are, in fact, three types of rules used in phonological analysis: 1) phonetic rules – for sounds within words; 2) phonemic rules – for variations of pronunciation when words are spoken together, and; 3) prosodic rules – for fluctuation in stress and intonation across a sentence.
- Morphology: This level deals with the componential nature of words, which are composed of morphemes – the smallest units of meaning.
- Lexical: At this level, humans, as well as NLP systems, interpret the meaning of individual words. Several types of processing contribute to word-level understanding – the first of these being assignment of a single part-of-speech tag to each word. In this processing, words that can function as more than one part-of-speech are assigned the most probable part-of-speech tag based on the context in which they occur.
- Syntactic: This level focuses on analyzing the words in a sentence so as to uncover the grammatical structure of the sentence.
- Semantic: Semantic processing determines the possible meanings of a sentence by focusing on the interactions among word-level meanings in the sentence. This level of processing can

include the semantic disambiguation of words with multiple senses; in an analogous way to how syntactic disambiguation of words that can function as multiple parts-of-speech is accomplished at the syntactic level. Semantic disambiguation permits one and only one sense of polysemous words to be selected and included in the semantic representation of the sentence.

- Discourse: While syntax and semantics work with sentence-length units, the discourse level of NLP works with units of text longer than a sentence. That is, it does not interpret multi-sentence texts as just concatenated sentences, each of which can be interpreted singly. Rather, discourse focuses on the properties of the text as a whole that convey meaning by making connections between component sentences.

- Pragmatic: This level is concerned with the purposeful use of language in situations and utilizes context over and above the contents of the text for understanding The goal is to explain how extra meaning is read into texts without actually being encoded in them. This requires much world knowledge, including the understanding of intentions, plans, and goals.

3.2 NLP Tasks

Natural language processing provides both theory and implementations for a range of applications. In fact, any application that utilizes text is a candidate for NLP. The most frequent applications utilizing NLP include the following:

- Information Retrieval (IR) & Web Search: the science of searching for documents, for information within documents, and for metadata about documents, as well as that of searching databases and the World Wide Web.

- Information Extraction (IE): focuses on the recognition, tagging, and extraction into a structured representation, certain key elements of information, e.g. persons, companies, locations, organizations, from large collections of text.

- Text Summarization: the process of distilling the most important information from a source in order to produce an abridged version.

- Question Answering (QA): the response from documents to extracted or generated answer.

- Machine Translation (MT): the use of computer software in order to translate text or speech from one natural language to another.

- Speech Recognition & Synthesis: the extraction of textual representation of a spoken utterance.

- Text Generation: A method for generating sentences from “keywords” or “headwords”.

- Natural Language Understanding and Generation (NLU, NLG): NLG system is like a translator that converts a computer based representation into a natural language representation.

3.2.1 Sentiment Analysis

Sentiment analysis, also called opinion mining, is the field of study that analyzes people’s opinions, sentiments, evaluations, appraisals, attitudes, and emotions towards entities such as products, services, organizations, individuals, issues, events, topics, and their attributes. It represents a large problem space [60]. Sentiment Analysis, in its simplest form, aims to detect *positive, neutral* or *negative* feelings from text.

Opinions are central to almost all human activities because they are key influencers of our behaviors. Whenever we need to make a decision, we want to know others’ opinions. In the real world, businesses and organizations always want to find consumer or public opinions about their products

and services. Individual consumers also want to know the opinions of existing users of a product before purchasing it, and others' opinions about political candidates before making a voting decision in a political election. In the past, when an organization or a business needed public or consumer opinions, it conducted surveys, opinion polls, and focus groups. Acquiring public and consumer opinions has long been a huge business itself for marketing, public relations, and political campaign companies.

With the explosive growth of social media (e.g., reviews, forum discussions, blogs, micro-blogs, Twitter, comments, and postings in social network sites) on the Web, individuals and organizations are increasingly using the content in these media for decision making. For an organization, it may no longer be necessary to conduct surveys, opinion polls, and focus groups in order to gather public opinions because there is an abundance of such information publicly available. However, finding and monitoring opinion sites on the Web and distilling the information contained in them remains a formidable task because of the proliferation of diverse sites. Each site typically contains a huge volume of opinion text that is not always easily deciphered in long blogs and forum postings. The average human reader will have difficulty identifying relevant sites and extracting and summarizing the opinions in them. Automated sentiment analysis systems are thus needed.

The levels on which sentiment analysis is performed are:

- Document level: The task at this level is to classify whether a whole opinion document expresses a positive or negative sentiment [61, 62]. For example, given a product review, the system determines whether the review expresses an overall positive or negative opinion about the product. This task is commonly known as document-level sentiment classification. This level of analysis assumes that each document expresses opinions on a single entity (e.g., a single product). Thus, it is not applicable to documents which evaluate or compare multiple entities.

- Sentence level: The task at this level goes to the sentences and determines whether each sentence expressed a positive, negative, or neutral opinion. Neutral usually means no opinion. This level of analysis is closely related to subjectivity classification [63], which distinguishes sentences (called objective sentences) that express factual information from sentences (called subjective sentences) that express subjective views and opinions.

- Entity and Aspect level: Both the document level and the sentence level analyses do not discover what exactly people liked and did not like. Aspect level performs finer-grained analysis. Instead of looking at language constructs (documents, paragraphs, sentences, clauses or phrases), aspect level directly looks at the opinion itself. It is based on the idea that an opinion consists of a sentiment (positive or negative) and a target (of opinion). An opinion without its target being identified is of limited use. Realizing the importance of opinion targets also helps us understand the sentiment analysis problem better. For example, although the sentence "although the service is not that great, I still love this restaurant" clearly has a positive tone, we cannot say that this sentence is entirely positive. In fact, the sentence is positive about the restaurant (emphasized), but negative about its service (not emphasized).

3.2.2 Emotion Recognition

Emotion Detection and Recognition from text is a recent field of research that is closely related to Sentiment Analysis. Emotion Analysis aims to detect and recognize types of feelings through the expression of texts, such as anger, disgust, fear, happiness, sadness, and surprise.

The Internet contains large collections of documents that can provide vast amounts of text. Sources of text useful for emotion analysis may come from product reviews, news articles, stock market analyses, personal blogs/journals, social network websites, forums, fiction excerpts, critiques, or political debates; anywhere that people discuss and share their opinion freely could be a source. Out of the many textual analysis tasks pursued, a growing area of interest is the automatic detection of emotion. Being an important element in understanding human experience and communication, emotions have

been studied in the psychological and behavioral sciences disciplines. Methods of emotion detection are now possible with the foundations of textual analysis; however, the lack of consistency in approaches creates challenges when trying to compare methods.

3.3 Language Modeling

Language models (LMs) compute the probability of occurrence of a number of words in a particular sequence. The probability of a sequence of T words $\{w_1, w_2, \ldots, w_T\}$ is denoted as $P(w_1, w_2, \ldots, w_T)$. The key goal of language modeling is to provide adequate probabilistic information so that the likely word sequences have a higher probability.

$$P(w_1, w_2, \ldots, w_T) = \prod_{i=1}^{i=T} P(w_i | w_1, w_2, \ldots, w_{i-1}) \tag{3.1}$$

3.3.1 N-gram Language Models

The simplest model that assigns probabilities to sentences and sequences of words is the *n-gram language model*. In n-gram language modeling, we break up the process of predicting a word sequence $w_1, w_2, \ldots, w_T$ into predicting one word at a time. To do this, we first decompose the probability using the chain rule:

$$P(w_1, w_2, \ldots, w_T) = P(w_1)P(w_2|w_1) \ldots P(w_T|w_1, w_2, \ldots, w_{T-1}) \tag{3.2}$$

Markov chain
The language model probability $P(w_1, w_2, \ldots, w_T)$ is a product of word probabilities given a history of preceding words. The intuition of the n-gram model is that instead of computing the probability of a word given its entire history, we can approximate the history by using only the last few words. So, to be able to estimate these word probability distribution, we limit the history to n words:

$$P(w_T|w_1, w_2, \ldots, w_{T-1}) \approx P(w_T|w_{T-n}, \ldots, w_{T-2}, w_{T-1}) \tag{3.3}$$

The model described in Equation 3.3, where we step through a sequence and consider for the transitions only a limited history, is called a *Markov chain* and the number of previous states (in our case, words) is the order of the model.

According to the *Markov assumption*, only a limited number of previous words affect the probability of the next word. While the kth order Markov assumption is clearly wrong for any k (sentences can have arbitrarily long dependencies), it still produces strong language modeling results for relatively small values of k, and was the dominant approach for language modeling for decades.

Typically, the actual number of words in the history is based on how much training data are available. Most commonly, *trigram* language models are used, which consider a two-word history to predict the third word. Language models may also be estimated over 2-grams (*bigrams*), single words (*unigrams*), or any other order of n-grams.

Estimation
For the most common case of trigram language model, the estimation of word prediction probabilities $P(w_3|w_1, w_2)$ is straightforward. We count how often in the training corpus the sequence w_1, w_2 is followed by the word w_3, as opposed to other words. According to maximum likelihood estimation, we compute:

$$P(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\sum_w \text{count}(w_1, w_2, w)} \tag{3.4}$$

Perplexity
There are several metrics for evaluating language modeling. The application-centric ones evaluate them in the context of performing a higher-level task, for example by measuring the improvement in translation quality when switching the language-modeling component in a translation system from model A to model B [64].

A more intrinsic evaluation of language models is using perplexity over unseen sentences. Perplexity is an information theoretic measurement of how well a probability model predicts a sample. Low perplexity values indicate a better fit. Given a text corpus of n words $w_1, w_2, \ldots, w_n$ and a language model function LM assigning a probability to a word based on its history, the perplexity of a LM with respect to the corpus is:

$$2^{-\frac{1}{n}\sum_{i=1}^{n}\log_2 \mathrm{LM}(w_i|w_{1:i-1})} \tag{3.5}$$

Good language models (i.e., reflective of real language usage) will assign high probabilities to the events in the corpus, resulting in lower perplexity values. The perplexity measure is a good indicator of the quality of a language model. Perplexities are corpus specific, so perplexities of two language models are only comparable with respect to the same evaluation corpus.

3.3.2 Neural Language Models

To overcome the shortcomings of traditional language models, non-linear neural network models have been proposed. They allow conditioning on increasingly large context sizes with only a linear increase in the number of parameters.

A neural probabilistic language model was popularized by Bengio et al. [7]. This model takes as input vector representations of a word window of n previous words, which are looked up in a table C. These vectors are now known as word embeddings. These word embeddings ($C(w) \in \mathbb{R}^{d_w}$) are then concatenated and fed into a hidden layer, whose output is provided to a softmax layer. So:

$$\begin{aligned} \boldsymbol{x} &= [C(w_1); C(w_2); \ldots; C(w_n)] \\ \hat{y} = P(w_i|w_{1:k}) = LM(w_{1:k}) &= \text{softmax}(\boldsymbol{h}\boldsymbol{W^2} + \boldsymbol{b^2}) \\ \boldsymbol{h} &= g(\boldsymbol{x}\boldsymbol{W^1} + \boldsymbol{b^1}) \\ \boldsymbol{x} &= [C(w_1); C(w_2); \ldots; C(w_n)] \\ C(w) &= \boldsymbol{E}_{[w]} \end{aligned} \tag{3.6}$$

where $w_i \in V$, $\boldsymbol{E} \in \mathbb{R}^{|V| \times d_w}$, $\boldsymbol{W^1} \in \mathbb{R}^{n \cdot d_w \times d_{\text{hid}}}$, $\boldsymbol{b^1} \in \mathbb{R}^{d_{\text{hid}}}$, $\boldsymbol{W^2} \in \mathbb{R}^{d_{\text{hid}} \times |V|}$, $\boldsymbol{b^2} \in \mathbb{R}^{|V|}$.

V is a finite vocabulary. The vocabulary size |V|, ranges between 1,000 - 1,000,000 words, with the common size being around 70,000 unique words.

More recently, feed-forward neural networks have been replaced with recurrent neural networks [65] and long short-term memory networks (LSTMs) [54] for language modeling.

3.4 Transfer Learning

Most NLP models, nowadays, rely on fixed distributional pretrained word representations, such as word2vec [8] and GloVe [66] to initialize their embedding layer. While such pretrained word vectors capture semantic similarities, they have limitations and cannot model context. Since they only assign one vector to each unique word, they cannot handle polysemy, figurative usage of language or a context that is highly different than the one they were pretrained on. Therefore, they cannot model the complexity and nuances of language. To this end, contextual pretrained word representations from language models have been proposed [25, 32], that assign a word vector to a word based on the context they find it in.

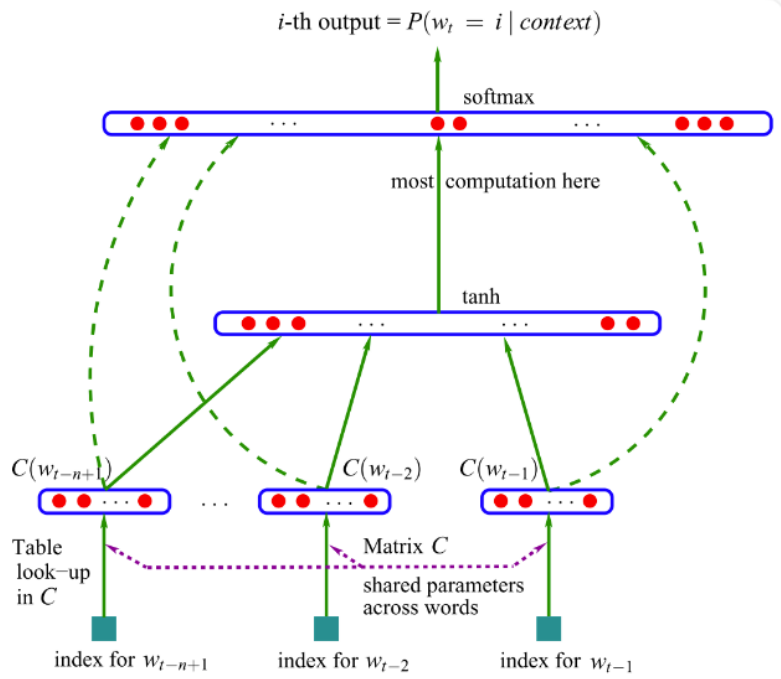


Figure 3.1: A feed-forward neural network language model. [7]

3.4.1 Word Embeddings

The key idea behind the unsupervised word vectors is that one would like the embedding vectors of "similar" words to have similar vectors. While word similarity is hard to define and is usually very task-dependent, current approaches derive from the *distributional hypothesis* [67], stating that words are similar if they appear in similar contexts. Different methods all create supervised training instances in which the goal is to either predict the word from its context, or predict the context from the word. Perhaps the most important set of pretrained embedding vectors is *word2vec*. Word2vec is an approximation of language modeling, applied to a fixed word window.

Word2vec [8]
Word2vec is a shallow, two-layer neural network which is trained to reconstruct linguistic contexts of words. It takes as its input a large corpus of words and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space. Word2Vec is a particularly computationally-efficient predictive model for learning word embeddings from raw text.

Given enough data, usage and contexts, word2vec can make highly accurate guesses about a word's meaning based on past appearances. Those guesses can be used to establish a word's association with other words (e.g. "king" is to "man" what "queen" is to "woman"), or cluster documents and classify them by topic. Those clusters can form the basis of search, sentiment analysis and recommendations in such diverse fields as scientific research, legal discovery, e-commerce and customer relationship management.

Word2Vec has two forms, the *Continuous Bag-of-Words* (CBOW) model and the *Skip-Gram* model, as illustrated in Figure 3.3. When the feature vector assigned to a word cannot be used to accurately predict that word's context, the components of the vector are adjusted. Each word's context in the corpus is the *teacher* sending error signals back to adjust the feature vector. The vectors of words judged similar by their context are nudged closer together by adjusting the numbers in the vector.

- *Continuous Bag of Words (CBOW)*

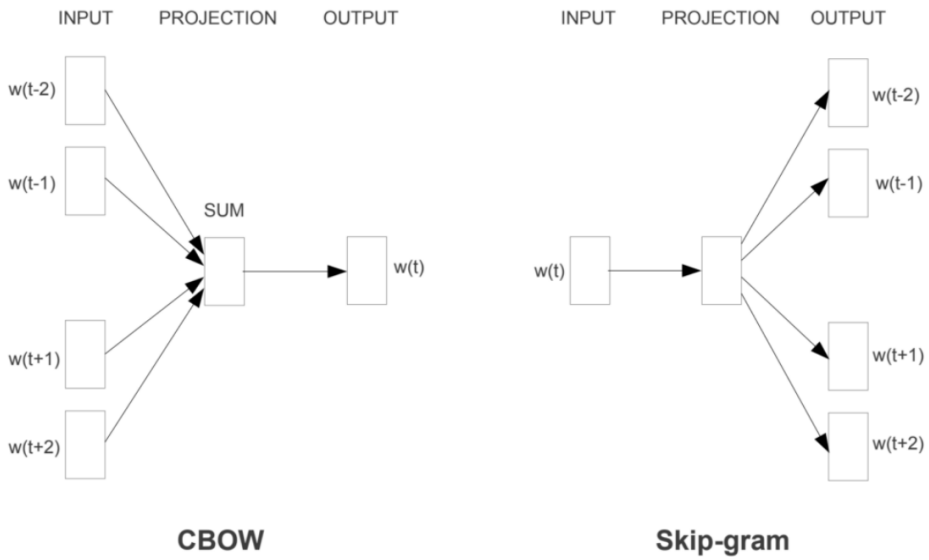


Figure 3.2: Word2vec training models. Taken from [8].

Supposing we want to predict word w_i, the input to the model could be $w_{i-2}, w_{i-1}, w_{i+1}, w_{i+2}$, the preceding and following words of the target word. The output of the neural network will be w_i. So, the CBOW model can be thought as learning word embeddings by training a model to predict a word given its context.

- *Skip-Gram*

  This model is the opposite of CBOW, as in this case the input of the model is w_i and the output would be $w_{i-2}, w_{i-1}, w_{i+1}, w_{i+2}$. The task, therefore, is to learn word embeddings by training a model to predict context given a word.

Even though the distributional hypothesis offers an appealing platform for deriving word similarities by representing words according to the contexts in which they occur, it has some inherent limitations, which should be taken into account when using the derived representations. The most important one, which has been largely examined in this thesis, is the lack of context.

Lack of context
The distributional approaches aggregate the contexts in which a term occurs in a corpus. The result is a context-free, or else *context-independent* word representation. An obvious problem that occurs is that polysemous words (words with obvious multiple senses) cannot be modeled properly. For example, a *bank* may refer to a financial institution or to the side of a river, a *star* may be an abstract shape, a celebrity or an astronomical entity, etc. By assigning the same vector to all the senses of a given word, language cannot be modeled in its complex form, as the meaning of numerous words evades.

Window of surrounding words
Another limitation comes from learning embeddings based only on a small window of surrounding words, sometimes words such as *good* and *bad* share almost the same embedding [68], which is problematic if used in tasks such as sentiment analysis [69]. At times these embeddings cluster semantically similar words which have opposing sentiment polarities. This leads the downstream model used for the sentiment analysis task to be unable to identify this contrasting polarities leading to poor performance.

3.4.2 Pretrained Word Representations from Language Models

Language models have recently been proposed to create word representations leveraging the context. Traditional word embedding methods such as word2vec consider all the sentences where a word is present in order to create a global vector representation of that word. However, a word can have completely different senses or meanings in the contexts. Since the language modeling task allows computing the joint probabilities over word sequences, it is an intuitively meaningful way of representing the context of a given word. Pretrained language models encode contextual information and high-level features of language, modeling syntax and semantics. They thus enable NLP models to better disambiguate meaning and understand the correct sense of a given word. The learned feature representation contains distilled knowledge of a certain language.

Moreover, language modeling relies on unlabeled data, which can be easily found and do not require extensive annotation. It is thus an appealing task, as it can leverage vast corpora that are available online. For instance, Universal Language Model Fine-Tuning (ULMFiT) [25] was pretrained on English Wikipedia.

Pretrained representations from language models can be used either as additional features ([32] or they can be directly fine-tuned to the target task [25, 24]. When they are used as additional features, task-specific architectures that include the pretrained representations as additional features are needed. The fine-tuning approach, on the other hand, such as the one adopted by the Generative Pretrained Transformer (OpenAI GPT) [26], requires far less training parameters and is trained on the downstream supervised learning tasks by simply fine-tuning the previous parameters. Moreover, in the case of OpenAI Transformer, uni-directional language models are used to model the perplexities of language. ELMo [32] and BERT [24], however, rely on bi-directional language models, in order to enhance the feature representations that are then used to tackle various tasks.

We will now briefly discuss two methods of leveraging word representations from language models to tackle downstream tasks, namely ULMFIT and ELMo.

1) Universal Language Model Fine-Tuning [25]

In this work, an effective transfer learning method that can be applied to any task in NLP was introduced. The proposed approach relies on pretraining a generic language model on a large unlabeled corpus (e.g. Wikipedia), fine-tuning it to the target dataset and then transferring it to a classification model and fine-tuning it until convergence.

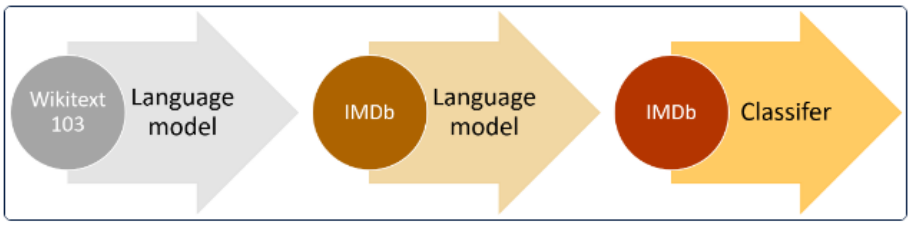


Figure 3.3: The three steps of ULMFiT.

First, the authors pretrain a state-of-the-art language model AWD-LSTM [70] on Wiki-103 with a vocabulary of approximately 300K words. The intuition is that an ImageNet-like corpus should be used for natural language, in order to capture is general properties.

Then, they further fine-tune the general-domain language model to the target dataset, which they treat as a corpus, removing the labels. They propose an elaborate fine-tuning scheme that relies on discriminative fine-tuning and slanted triangular learning rate.

Discriminative fine-tuning

They propose a novel fine-tuning method, *discriminative fine-tuning*, building on the premise that as different layers capture different types of information [71], they should not be fine-tuned to the same degree. So, instead of using the regular stochastic gradient descent (2.25), they split the parameters θ

into $\{\theta^1, \ldots, \theta^L\}$, where θ^l contains the parameters of the model at the l-th layer and L is the number of layers of the model. Similarly, the learning rate of each layer is given by $\{\eta^1, \ldots, \eta^L\}$, where η^l is the learning rate of the l-th layer. So, the SGD update with discriminative fine-tuning obtains the following form:

$$\theta_t^l = \theta_{t-1}^l - \eta^l \nabla_{\theta^l} J(\theta) \tag{3.7}$$

Slanted triangular learning rates

They also propose *slanted triangular learning rates* (STLR), a method that first linearly increases the learning rate and then linearly decays it. The intuition is that for adapting the parameters of the language model to task-specific features, it is desirable that a model quickly converges to a suitable region of the parameter space in the beginning of training and then refines its parameters. STLR has the following form:

$$cut = [T \cdot cut\_frac] \tag{3.8}$$

$$\mathrm{p} = \begin{cases} \frac{t}{cut}, & \text{if } t < cut \\ 1 - \frac{t-cut}{cut \cdot (\frac{1}{cut\_frac} - 1)}, & \text{otherwise} \end{cases} \tag{3.9}$$

$$\eta_t = \eta_{max} \cdot \frac{1 + p \cdot (ratio - 1)}{ratio} \tag{3.10}$$

where T is the number of training iterations, cut_frac is the fraction of iterations they increase the learning rate, cut is the iteration when they switch from increasing to decreasing the learning rate, p is the fraction of the number of iterations they have increased or will decrease the learning rate respectively. $ratio$ specifies how much smaller the lowest learning rate is from the maximum learning rate η_{max}, and η_t is the learning rate at iteration t. STLR is a modification of triangular learning rates [72].

Finally, the authors transfer the pretrained language model to a classification model, augmenting it with two additional linear blocks and using *gradual unfreezing* to prevent overfitting, which basically relies in first unfreezing the task-specific layer and fine-tune it for one epoch, then unfreeze the next lower layer until all layers have been fine-tuned.

The proposed method achieves competitive performance, reducing the test error rates in a variety of classification datasets. It requires, though, careful fine-tuning and uses sophisticated techniques to adjust the source task distribution to the target task distribution.

2) Embeddings from Language Models (ELMo) [32]

ELMo embeddings are deep contextualized word representations that offer high quality representations for language, modeling both complex characteristics of word use and adjusting them in different linguistic contexts. The vectors are derived from a bi-directional LSTM that is trained with a coupled language model (LM) objective on a large text corpus. ELMo representations are a function of all the internal layers of the bi-directional language model. They are used concatenated with pretrained word vectors (such as word2vec, GloVe) to augment the information captured and result in state-of-the-art results in different downstream tasks.

Given that for each token t_k, $\boldsymbol{x}_k^{LM}$ is a context-independent token representation (via token embeddings or a convolutional neural network (CNN) over characters), a L-layer bi-directional LSTM (2.5.3) computes a set of $2L + 1$ representations

$$\begin{aligned} R_k &= \{\boldsymbol{x}_k^{LM}, \overrightarrow{\boldsymbol{h}}_{k,j}^{LM}, \overleftarrow{\boldsymbol{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\ &= \boldsymbol{h}_{k,j}^{LM} \mid j = 0, \ldots, L\} \end{aligned} \tag{3.11}$$

ELMo embeddings are given by a task-specific weighting of the intermediate layers of a pretrained language model:

$$\text{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \boldsymbol{h}_{k,j}^{LM} \tag{3.12}$$

where s^{task} are softmax-normalized weights and the scalar parameter γ^{task} allows the task model to scale the entire ELMo vector.

ELMo embeddings are then added to six benchmark datasets that include question answering, textual entailment, semantic role labeling, coreference resolution, named entity extraction and sentiment analysis and establish new state-of-the-art results in all of them. However, the weighting of the ELMo embeddings needs to be carefully tuned for every different task. Therefore, the proposed method, although very effective, has some limitations and requires task-specific architectures.

Chapter 4

Ensemble of Neural Transfer Learning Methods for Implicit Emotion Classification

4.1 Introduction

Emotion recognition is a particularly interesting problem in NLP, as language usually reflects the emotional state of an individual. Therefore, it is natural to study human emotions by understanding how they are reflected in text. The task of automatically recognizing emotions from text has received lots of attention from the scientific community. This task is in most cases formally defined as the classification of words, phrases, or documents into a number of predefined emotion categories or dimensions. In some cases, regression problems can also be formulated, as for example the task of predicting the degree to which an emotion is expressed in text [73].

Social media, especially micro-blogging services like Twitter, have attracted lots of attention from the NLP community. The language used is constantly evolving by incorporating new syntactic and semantic constructs, such as emojis or hashtags, abbreviations and slang, making natural language processing in this domain even more demanding. Moreover, the analysis of such content leverages the high availability of datasets offered from Twitter, satisfying the need for large amounts of data for training. Emotion recognition is particularly interesting in social media, as it has useful applications in numerous tasks, such as public opinion detection about political tendencies [74, 75, 76], stock market monitoring [77, 78], tracking product perception [79], even detection of suicide-related communication [80].

In the past, emotion analysis, like most NLP tasks, was tackled by traditional methods that included hand-crafted features or features from sentiment lexicons [81, 82, 83] which were fed to classifiers such as Naive Bayes and SVMs [84, 85, 86]. However, deep neural networks achieve increased performance compared to traditional methods, due to their ability to learn more abstract features from large amounts of data, producing state-of-the-art results in emotion recognition and sentiment analysis [87, 88, 89].

In this chapter, we present our work to approach a multi-class classification task. The task consists in predicting the emotion of a word excluded from a tweet. Removed words (or *trigger-words*) include the terms “sad”, “happy”, “disgusted”, “surprised”, “angry”, “afraid” and their synonyms. Given a tweet, we are asked to predict the emotion it conveys, specifically anger, disgust, fear, joy, sadness and surprise.

We address this implicit emotion classification task (Implicit Emotion recognition Shared Task - IEST [90]) leveraging abstract pretrained feature representations from models that have been trained on different tasks. Transfer learning offers a meaningful starting point in order to improve the performance of a model on the given task. Our proposed model employs 3 different transfer learning schemes of pretrained models: word embeddings, a sentiment model and language models.

4.2 Related Work

Transfer Learning (TL) uses knowledge from a learned task so as to improve the performance of a related task by reducing the required training data [5, 91]. In computer vision, transfer learning is em-

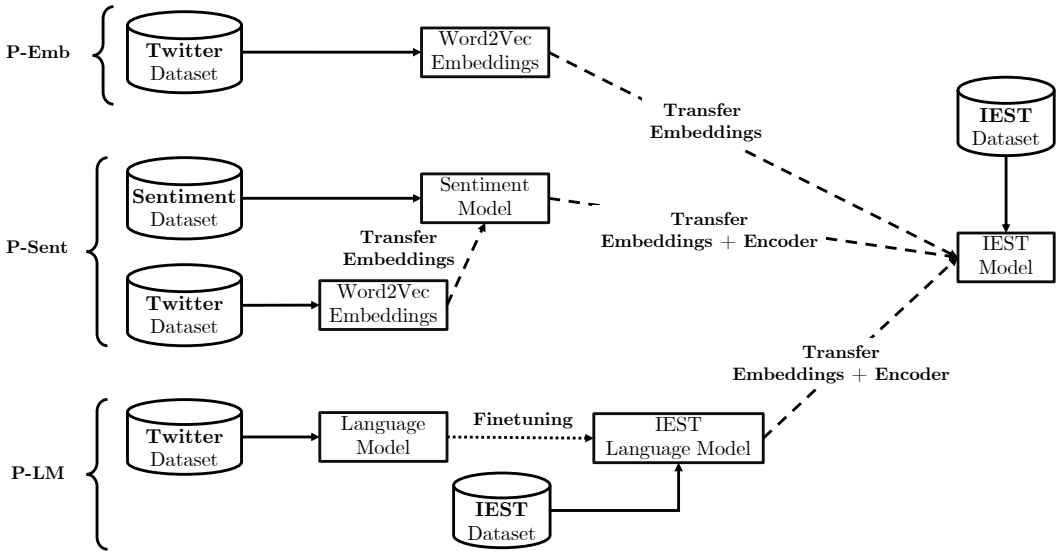


Figure 4.1: High-level overview of our transfer learning approaches.

ployed in order to overcome the deficit of training samples for some categories by adapting classifiers trained for other categories [92]. Notable examples include face recognition [93] and visual question-answering [94], where image features trained on ImageNet [14] and word embeddings estimated on large corpora via unsupervised training are combined.

Following this logic, transfer learning methods have also been applied to NLP. Pretrained word vectors [95, 66] have become standard components of most architectures. Prerained word representations have shown to improve models for entailment [96], sentiment analysis [97], summarization [98], and question answering [99].

To obtain high quality word vectors, though, we need to introduce deep contextualized word representations. These representations should address the challenge of modeling complex characteristics of word use, such as syntax and semantics, and use them appropriately for different linguistic contexts. Such vectors handle polysemy and encode the nuances of language.

Recently, approaches that leverage pretrained language models have emerged, which learn the compositionality of language, capture long-term dependencies and context-dependent features. For instance, ELMo contextual word representations [32] and ULMFiT [100] achieve state-of-the-art results on a wide variety of NLP tasks. Our work is mainly inspired by ULMFiT, which we extend to the Twitter domain.

4.3 Proposed Model

All of our transfer learning schemes share the same architecture: A 2-layer LSTM with a self-attention mechanism. It is shown in Figure 4.2.

Embedding Layer
The input to the network is a Twitter message, treated as a sequence of words. We use an embedding layer to project the words $w_1, w_2, ..., w_N$ to a low-dimensional vector space R^W, where W is the size of the embedding layer and N the number of words in a tweet.

LSTM Layer
An LSTM takes as input a sequence of word embeddings and produces word annotations $h_1, h_2, ..., h_N$, where h_i is the hidden state at time-step i, summarizing all the information of the sentence up to w_i. We use bidirectional LSTM to get word annotations that summarize the information from both directions. A bi-LSTM consists of a forward $\overrightarrow{f}$ that parses the sentence from w_1 to w_N and a backward $\overleftarrow{f}$ that parses it from w_N to w_1. We obtain the final annotation for each word h_i, by concatenating

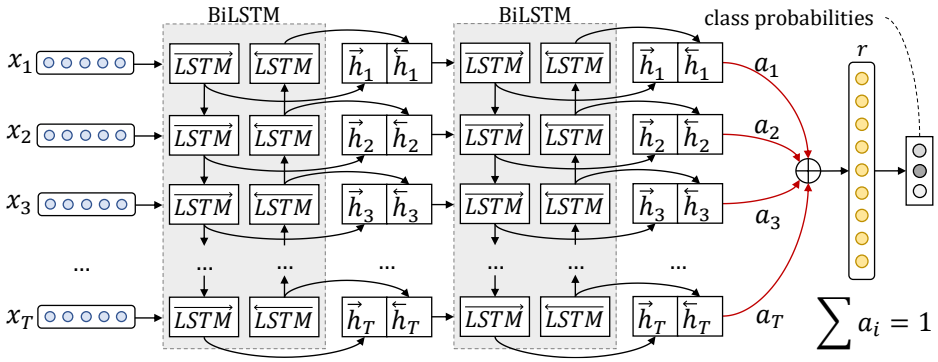


Figure 4.2: The proposed model, composed of a 2-layer bi-LSTM with a deep self-attention mechanism. When the model is initialized with pretrained LMs, we use uni-directional LSTM instead of bi-directional.

the annotations from both directions, $h_i = \overrightarrow{h_i} \parallel \overleftarrow{h_i}, \quad h_i \in R^{2L}$, where $\parallel$ denotes the concatenation operation and L the size of each LSTM. When the network is initialized with pretrained LMs, we employ unidirectional instead of bi-LSTMs.

Attention Layer

To amplify the contribution of the most informative words, we augment our LSTM with an attention mechanism, which assigns a weight a_i to each word annotation h_i. We compute the fixed representation r of the whole input message, as the weighted sum of all the word annotations.

$$e_i = tanh(W_h h_i + b_h), \quad e_i \in [-1, 1] \tag{4.1}$$

$$a_i = \frac{exp(e_i)}{\sum_{t=1}^{T} exp(e_t)}, \quad \sum_{i=1}^{T} a_i = 1 \tag{4.2}$$

$$r = \sum_{i=1}^{T} a_i h_i, \quad r \in R^{2L} \tag{4.3}$$

where W_h and b_h are the attention layer's weights.

Output Layer

We use the representation r as feature vector for classification and we feed it to a fully-connected softmax layer with L neurons, which outputs a probability distribution over all classes p_c as described in Eq. 4.4:

$$p_c = \frac{e^{Wr+b}}{\sum_{i \in [1,L]} (e^{W_i r + b_i})} \tag{4.4}$$

where W and b are the layer's weights and biases.

4.3.1 Pretrained Word Embeddings

In the first approach, we train *word2vec* word embeddings with which we initialize the embedding layer of our network. The weights of the embedding layer remain frozen during training. The *word2vec* word embeddings are trained on the 550M Twitter corpus, with negative sampling of 5 and minimum word count of 20, using Gensim's [101] implementation. The resulting vocabulary contains $800,000$ words.

4.3.2 Pretrained Classifier

In the second approach, we first train a sentiment analysis model on the sentiment analysis (Sent 17) dataset, using the architecture described in Sec. 4.3. The embedding layer of the network is initialized with our pretrained word embeddings. Then, we fine-tune the network on the IEST task, by replacing its last layer with a task-specific layer.

4.3.3 Pretrained Language Model

The third approach consists of the following steps: (1) we first train a language model on a generic Twitter corpus, (2) we fine-tune the LM on the task at hand and finally, (3) we transfer the embedding and RNN layers of the LM, we add attention and output layers and fine-tune the model on the target task.

LM Pretraining

We collect three Twitter datasets as described in Sec. 4.4.1 and for each one we train an LM. In each dataset we use the 50,000 most frequent words as our vocabulary. Since the literature concerning LM transfer learning is limited, especially in the Twitter domain, we aim to explore the desired characteristics of the pretrained LM. To this end, our contribution in this research area lies in experimenting with a task-relevant corpus (EmoCorpus), a generic one (GenCorpus) and a mixture of both (EmoCorpus+).

LM Fine-tuning

This step is crucial since, albeit the diversity of the general-domain data used for pretraining, the data of the target task will likely have a different distribution.

We thus fine-tune the three pretrained LMs on the IEST dataset, employing two approaches. The first is simple fine-tuning, according to which all layers of the model are trained simultaneously. The second one is a simplified yet similar approach to *gradual unfreezing*, proposed in [100], which we denote as *Sequential Unfreezing* (SU). According to this method, after we have transfered the pretrained embedding and LSTM weights, we let only the output layer fine-tune for $n-1$ epochs. At the n^{th} epoch, we unfreeze both LSTM layers. We let the model fine-tune, until epoch $k-1$. Finally, at epoch k, we also unfreeze the embedding layer and let the network train until convergence. In other words, we experiment with pairs of numbers of epochs, {n, k}, where n denotes the epoch when we unfreeze the LSTM layers and k the epoch when we unfreeze the embedding layer. Naive fine-tuning poses the risk of catastrophic forgetting, or else abruptly losing the knowledge of a previously learnt task, as information relevant to the current task is incorporated. Therefore, to prevent this from happening, we unfreeze the model starting from the last layer, which is task-specific, and after some epochs we progressively unfreeze the next, more general layers, until all layers are unfrozen. Our unfreezing method is depicted in Figure 4.3.

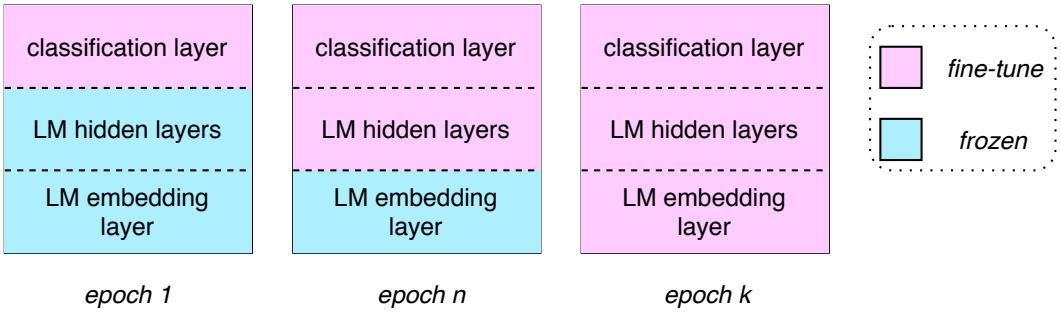


Figure 4.3: Sequential unfreezing. We unfreeze the hidden layers of the LM at epoch n, and we also unfreeze the embedding layer of the LM at epoch k.

LM Transfer

This is the final step of our TL approach. We now have several LMs from the second step of the procedure. We transfer their embedding and RNN weights to a final target classifier. We again experiment with both simple and more sophisticated fine-tuning techniques, to find out which one is more helpful to this task.

Furthermore, we introduce the *concatenation method*, which was inspired by the correlation of language modeling and the task at hand. We use pretrained LMs to leverage the fact that the task is basically a cloze test. In an LM, the probability of occurrence of each word, is conditioned on the preceding context, $P(w_t|w_1, \ldots, w_{t-1})$. In RNN-based LMs, this probability is encoded in the hidden state of the RNN, $P(w_t|h_{t-1})$. To this end, we concatenate the hidden state of the LSTM, right before the missing word, $h_{implicit}$, to the output of the self-attention mechanism, r:

$$r' = r \parallel h_{implicit}, \quad h_i \in R^{2L} \tag{4.5}$$

where L is the size of each LSTM, and then feed it to the output linear layer. This way, we preserve the information which implicitly encodes the probability of the missing word.

4.3.4 Ensembling

We combine the predictions of our 3 transfer learning schemes with the intent of increasing the generalization ability of the final classifier. To this end, we employ a pretrained word embeddings approach, as well as a pretrained sentiment model and a pretrained LM. We use two ensemble schemes, namely unweighted average and majority voting.

Unweighted Average (UA)
In this approach, the final prediction is estimated from the unweighted average of the posterior probabilities for all different models. Formally, the final prediction p for a training instance is estimated by:

$$p = \arg\max_c \frac{1}{C} \sum_{i=1}^{M} \vec{p_i}, \quad p_i \in \mathbb{R}^C \tag{4.6}$$

where C is the number of classes, M is the number of different models, $c \in \{1, ..., C\}$ denotes one class and $\vec{p_i}$ is the probability vector calculated by model $i \in \{1, ..., M\}$ using softmax function.

Majority Voting (MV)
Majority voting approach counts the votes of all different models and chooses the class with most votes. Compared to UA, MV is affected less by single-network decisions. However, this schema does not consider any information derived from the minority models. Formally, for a task with C classes and M different models, the prediction for a specific instance is estimated as follows:

$$\begin{aligned} v_c &= \sum_{i=1}^{M} F_i(c) \\ p &= \arg\max_{c \in \{1,\ldots,C\}} v_c \end{aligned} \tag{4.7}$$

where v_c denotes the votes for class c from all different models, F_i is the decision of the i^{th} model, which is either 1 or 0 with respect to whether the model has classified the instance in class c or not and p is the final prediction.

4.4 Experiments & Results

4.4.1 Experimental Dataset

Apart from the IEST dataset, we employ a SemEval dataset for sentiment classification and other manually-collected unlabeled corpora for our language models.

Unlabeled Twitter Corpora
We collected a dataset of 550 million archived English Twitter messages, from 2014 to 2017. This dataset is used for calculating word statistics for our text preprocessing pipeline and training our *word2vec* word embeddings presented in Sec. 4.3.1.

For training our language models, described in Sec. 4.3.3, we sampled three subsets of this corpus. The first consists of 2 million (2M) tweets, all of which contain emotion words. To create the dataset, we selected tweets that included one of the six emotion classes of our task (*anger, disgust, fear, joy, sadness* and *surprise*) or synonyms. We ensured that this dataset is balanced by concatenating approximately 350K tweets from each category. The second chunk has 5M tweets, randomly selected from the initial 550M corpus. We aimed to create a general sub-corpus, so as to focus on the structural relationships of words, instead of their emotional content. The third chunk is composed of the two aforementioned corpora. We concatenated the 2 million emotion dataset with 2 million generic tweets, creating a final 4 million (4M) dataset. We denote the three corpora as *EmoCorpus* (2M), *EmoCorpus*+ (4M) and *GenCorpus* (5M).

Sentiment Analysis Dataset
We use the dataset of SemEval17 Task4A (Sent17) [102] for training our sentiment classifier as described in Sec. 4.3.2. The dataset consists of Twitter messages annotated with their sentiment polarity (*positive, negative, neutral*). The training set contains 56K tweets and the validation set 6K tweets.

4.4.2 Experimental Setup

Training
We use Adam algorithm [103] to optimize our networks, with mini-batches of size 64 and clip the norm of the gradients [104] at 0.5, as an extra safety measure against exploding gradients. We also used PyTorch [105] and Scikit-learn [106].

Hyperparameters
For all our models, we employ the same 2-layer attention-based LSTM architecture (Sec. 4.3). All the hyperparameters used are shown in Table 4.1.

Layer	P-Emb	P-Sent	P-LM
Embedding	300	300	400
Embedding noise	0.1	0.1	0.1
Embedding dropout	0.2	0.2	0.2
LSTM size	400	400	600/800
LSTM dropout	0.4	0.4	0.4

Table 4.1: Hyper-parameters of our models.

4.4.3 Results

Baselines
In Table 4.3 we compare the proposed transfer learning approaches against two strong baselines: (1) a Bag-of-Words (BoW) model with TF-IDF weighting and (2) a Bag-of-Embeddings (BoE) model,

where we retrieve the *word2vec* representations of the words in a tweet and compute the tweet representation as the centroid of the constituent *word2vec* representations. Both *BoW* and *BoE* features are then fed to a linear SVM classifier, with tuned $C = 0.6$. All of our reported F1-scores are calculated on the evaluation (*dev*) set, due to time constraints.

P-Emb and P-Sent models (4.3.1, 4.3.2)

We evaluate the *P-Emb* and *P-Sent* models, using both bi-directional and uni-directional LSTMs. The F1 score of our ensembling models is shown in Table 4.3. As expected, bi-LSTM models achieve higher performance.

LM Fine-tuning	LM Transfer			
	Simple fine-tuning	Sequential Unfreezing	Concat.	F1
Simple fine-tuning	✓			0.672
	✓		✓	0.667
		✓		0.676
		✓	✓	0.673
Sequential Unfreezing	✓			0.673
	✓		✓	0.667
		✓		0.678
		✓	✓	**0.682**

Table 4.2: Results of the P-LM, pretrained on the EmoCorpus. The first column refers to the fine-tuning procedure followed in the *LM Fine-tuning* step, while the second column describes the fine-tuning way used in *LM Transfer* step. Concat. stands for our concatenation method.

P-LM (4.3.3)

For the experiments with the pretrained LMs, we intend to transfer not just the first layer of our network, but rather the whole model, so as to capture more high-level features of language. As mentioned above, there are three distinct steps concerning the training procedure of this TL approach: (1) *LM pretraining*: we train three LMs on the EmoCorpus, EmoCorpus+ and GenCorpus corpora, (2) *LM fine-tuning*: we fine-tune the LMs on the IEST dataset, with 2 different ways. The first one is simple fine-tuning, while the second one is our sequential unfreezing (SU) technique. (3) *LM transfer*: We now have 6 LMs, fine-tuned on the IEST dataset. We transfer their weights to our final emotion classifier, we add attention to the LSTM layers and we experiment again with our 2 ways of fine-tuning and the *concatenation method* proposed in Sec. 4.3.3.

When trained on EmoCorpus, the P-LM model has a F1-score 68.2%. When trained on EmoCorpus+, F1-score is equal to 68.0%. Finally, when trained on GenCorpus, a F1-score of 67.5% is obtained. Even though EmoCorpus contains less training examples, *P-LM*s trained on it learn to encode more useful information for the task at hand.

In Table 4.2 we present all possible combinations of transferring the *P-LM* to the IEST task. We observe that sequential unfreezing consistently outperforms simple fine-tuning. Due to the difficulty in running experiments for all possible combinations, we compare our best approach, namely *SU + Concat.*, with *P-LM*s trained on our three unlabeled Twitter corpora.

Our submitted model is an ensemble of the models with the best performance. More specifically, we leverage the following models: (1) transfer learning of pretrained word embeddings, (2) transfer learning of pretrained sentiment classifier, (3) transfer learning of 3 different LMs, trained on 2M, 4M and 5M respectively. We use Unweighted Average (UA) ensembling of our best models from all aforementioned approaches. Our final results on the evaluation data are shown in Table 4.3.

Model	F1
Bag of Words (BoW)	60.1
Bag of Embeddings (BoE)	60.5
Ensembling (UA) P-Emb + P-Sent	68.4
Ensembling (UA) P-Sent + P-LM	69.5
Ensembling (UA) P-Emb + P-LM	70.1
Ensembling (MV) All	70.0
Ensembling (UA) All	**70.2**

Table 4.3: Results of our experiments when tested on the evaluation (*dev*) set. *BoW* and *BoE* are our baselines, while *P-Emb*, *P-Sent* and *P-LM* our proposed TL approaches. *UA* stands for Unweighted Average and *MV* for Majority Voting ensembling.

4.4.4 Discussion

As shown in Table 4.3, we observe that all of our proposed models achieve individually better performance than our baselines by a large margin. As far as the ensembling is concerned, both approaches, *MV* and *UA*, yield similar performance improvement over the individual models. In particular, we notice that adding the *P-LM* predictions to the ensemble contributes the most. This indicates that *P-LMs* encode more diverse information compared to the other approaches.

We also conduct an ablation study. The results are shown in Table 4.4. We notice that, when the three models are trained with uni-directional LSTM and the same number of parameters, the *P-LM* outperforms both the *P-Emb* and the *P-Sent* models. As expected, the upgrade to bi-LSTM improves the results of *P-Emb* and *P-Sent*. It could be the case that the augmented feature representation captured by a bi-directional layer allows the model to make better predictions about the emotion of a given sentence. We hypothesize that *P-LM* with bi-directional pretrained language models would have outperformed both of them. Furthermore, we conclude that both sequential unfreezing for fine-tuning and the concatenation method enhance the performance of the *P-LM* approach. As far as sequential unfreezing is concerned, it permits smooth iterative training of the layers of our model. Therefore, it preserves the knowledge encoded in the source task and it fine-tunes gradually the layers, starting from the output layer (task-specific) and proceeding to the hidden layers and finally the embedding layer (generic).

TL Model	F1	TL Model	F1	TL Model	F1
P-Emb	66.8	P-Sent	67.1	P-LM	67.5
P-Emb + bidir.	**68.4**	P-Sent + bidir.	**67.4**	P-LM + SU	67.9
P-Emb + SU	65.4	P-Sent + SU	66.5	P-LM + SU + Concat.	**68.2**
P-Emb + SU + Concat.	66.4	P-Sent + SU + Concat.	66.8		

Table 4.4: Ablation study of our three proposed transfer learning approaches, namely *P-Emb*, *P-Sent* and *P-LM*. *SU* stands for Sequential Unfreezing, *bidir.* for bi-LSTM, *Concat.* for the concatenation method.

4.5 Conclusions

In this work, we describe our deep-learning methods for missing emotion words classification, in the Twitter domain. The proposed approach is based on an ensemble of Transfer Learning techniques. We demonstrate that the use of refined, high-level features of text, as the ones encoded in language models, yields a higher performance. In the future, we aim to experiment with subword-level models, as they

have shown to consistently face the out-of-vocabulary (OOV) words problem [107, 108], which is more evident in Twitter. Moreover, we would like to explore other transfer learning approaches.

Finally, we share the source code of our models [1], in order to make our results reproducible and facilitate further experimentation in the field.

[1] /github.com/alexandra-chron/wassa-2018

Chapter 5

Transfer Learning from Language Models using an auxiliary objective

5.1 Introduction

In Natural Language Processing, distributed representations offered by pretrained word vectors such as word2vec [8] and GloVe [66] have become common initializations for deep learning models. Distributed word vectors have shown to effectively model word similarities and thus provide a meaningful starting point for deep learning. Transferring information from large amounts of unlabeled training data has shown to boost performance over random initialization of models in a variety of downstream tasks, such as part-of-speech tagging [109], named entity recognition [66] and question answering [110]. However, context cannot be model adequately by these pretrained word vectors, since they typically assign a vector to each word. Polysemy, thus, cannot be handled and a certain word's meaning sometimes evades, as each word only has a fixed vector representation.

Pretrained word representations captured by Language Models (LMs) have recently become popular in NLP. Pretrained LMs encode contextual information and high-level features of language, modeling syntax and semantics, producing state-of-the-art results across a wide range of tasks, such as named entity recognition [27], machine translation [111] and text classification [25]. They assign a different vector to each occurrence of a given word, based on its neighbors. So, they adapt to domain change and provide rich contextual word embeddings.

However, in cases where contextual embeddings from language models are used as additional features (e.g. ELMo [32]), results come at a high computational cost and require task-specific architectures. At the same time, approaches that rely on fine-tuning a LM to the task at hand (e.g. ULMFiT [25]) depend on pretraining the model on an extensive vocabulary and on employing a sophisticated slanted triangular learning rate scheme to adapt the parameters of the LM to the target dataset.

We propose a simple and effective transfer learning approach, that leverages LM contextual representations and does not require any elaborate scheduling schemes during training. We initially train a LM on a Twitter corpus and then transfer its weights. We add a task-specific recurrent layer and a classification layer. The transferred model is trained end-to-end using an auxiliary LM loss, which allows us to explicitly control the weighting of the pretrained part of the model and ensure that the distilled knowledge it encodes is preserved.

Our contributions are summarized as follows:

- We show that transfer learning from language models can achieve competitive results, while also being intuitively simple and computationally effective.
- We address the problem of catastrophic forgetting, by adding an auxiliary LM objective and using an unfreezing method.
- Our results show that our approach is competitive with more sophisticated transfer learning methods.

5.2 Related Work

Unsupervised pretraining has played a key role in deep neural networks, building on the premise that representations learned for one task can be useful for another task. In NLP, pretrained word vectors [95, 66] are widely used, improving performance in various downstream tasks, such as part-of-speech tagging [109] and question answering [110].

Aiming to learn from unlabeled data, Dai et al. [28] use unsupervised objectives such as sequence autoencoding and language modeling as pretraining representations. Ramachandran et al. [111] also pretrain encoder-decoder pairs using language models and fine-tune them to a specific task. Embeddings from Language Models (ELMo) embeddings [32] are obtained from character-based bidirectional language models improving the results in a variety of tasks as additional contextual representations.

Towards the same direction, Universal Language Model Fine-Tuning (ULMFiT) [25] shows impressive results on a variety of tasks by employing pretrained LMs. The proposed pipeline requires three distinct steps, that include pretraining the LM, fine-tuning it on a target dataset with an elaborate scheduling procedure and transferring it to a classification model.

Multi-Task Learning (MTL) via hard parameter sharing [112] in neural networks has proven to be effective in many NLP problems [113]. More recently, alternative approaches have been suggested that only share parameters across lower layers [114]. By introducing part-of-speech tags at the lower levels of the network, the proposed model achieves competitive results on chunking and Combinatory Category Grammar (CCG) super tagging. Our auxiliary language model objective follows this line of thought and intends to boost the performance of the higher classification layer.

5.3 Proposed Model

We introduce **SiATL**, which stands for **Si**ngle-step **A**uxiliary loss **T**ransfer **L**earning. In our proposed approach, we first train a LM. We then transfer its weights and add a task-specific recurrent layer to the final classifier. We also employ an auxiliary LM loss to avoid catastrophic forgetting.

5.3.1 Unsupervised Pretraining

We train a word-level language model, which consists of an embedding Long Short-Term Memory (LSTM) layer [53], 2 hidden LSTM layers and a linear layer. We want to minimize the negative log-likelihood of the LM:

$$L(\hat{p}) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T^n} \log \hat{p}(x_t^n | x_1^n, ..., x_{t-1}^n) \tag{5.1}$$

where $\hat{p}(x_t^n | x_1^n, ..., x_{t-1}^n)$ is the distribution of the t^{th} word in the n^{th} sentence given the $t-1$ words preceding it and N is total number of sentences.

5.3.2 Transfer & Auxiliary Loss

We transfer the weights of the pretrained model and add one LSTM with a self-attention mechanism [115, 13].

In order to adapt the contribution of the pretrained model to the task at hand, we introduce an auxiliary LM loss during training. The joint loss is the weighted sum of the task-specific loss L_{task} and the auxiliary LM loss L_{LM}, where γ is a weighting parameter to enable adaptation to the target task but at the same time keep the useful knowledge from the source task. Specifically:

$$L = L_{task} + \gamma L_{LM} \tag{5.2}$$

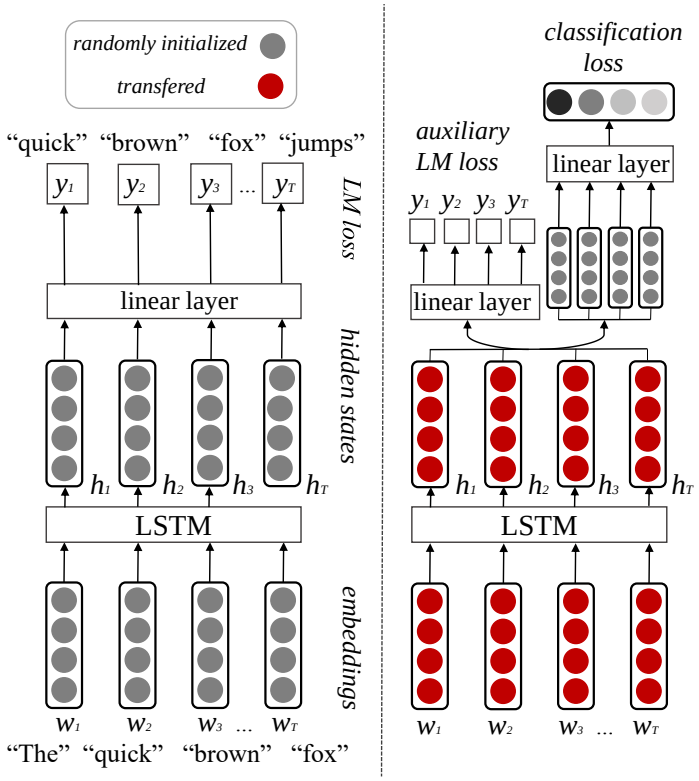


Figure 5.1: High-level overview of our proposed TL architecture. We transfer the pretrained LM add an extra recurrent layer and an auxiliary LM loss.

5.3.3 Exponential Decay of Joint Loss Weighting Factor

An advantage of the proposed transfer learning method is that the contribution of the LM can be explicitly controlled in each training epoch. In the first few epochs, the LM should contribute more to the joint loss of SiATL so that the task-specific layers adapt to the new linguistic distribution. After the knowledge of the pretrained LM is transferred to the new domain, the task-specific component of the loss function is more important and joint loss weighting factor (γ) should become smaller. In this paper, we use an exponential decay for the γ over the training epochs.

5.3.4 Sequential Unfreezing

Instead of fine-tuning all the layers simultaneously, we propose unfreezing them sequentially, according to Chronopoulou et al. [2]. We first fine-tune only the extra, randomly initialized LSTM and the output layer for $n-1$ epochs. At the n^{th} epoch, we unfreeze the pretrained hidden layers. We let the model fine-tune, until epoch $k-1$. Finally, at epoch k, we also unfreeze the embedding layer and let the network train until convergence. The values of n and k are obtained through hyperparameter tuning. We find the sequential unfreezing scheme important, as it minimizes the risk of overfitting to small datasets.

5.3.5 Optimizers

We use Stochastic Gradient Descent (SGD) for the pretrained LM with a small learning rate, in order to preserve its contextual information and avoid catastrophic forgetting. However, we want the extra LSTM and softmax layer to train fast and adapt to the target task, so in that case Adam [50] is employed.

5.4 Experiments & Results

5.4.1 Experimental Dataset

To pretrain the language model, we collect a dataset of 20 million English Twitter messages, including approximately 2M unique tokens. We use the 70K most frequent tokens as vocabulary. We evaluate our model on five datasets: *Sent17* for sentiment analysis [102], *PsychExp* for emotion recognition [116], *Irony18* for irony detection [117], *SCv1* and *SCv2* for sarcasm detection [118, 119]. More details about the datasets can be found in Table 5.1.

Dataset	Domain	# classes	# training data	# test data
Irony18	Tweets	4	3834	784
Sent17	Tweets	3	49570	12284
SCv2	Debate Forums	2	1000	2260
SCv1	Debate Forums	2	1000	995
PsychExp	Experiences	7	1000	6480

Table 5.1: Datasets used for the downstream tasks.

5.4.2 Experimental Setup

To preprocess the tweets, we use *Ekphrasis* [89]. For the generic datasets, we use NLTK [120]. For the Neural Bag-of-Words (NBoW) baseline, we use *word2vec* [95] 300-dimensional embeddings as features. For the neural models, we use an LM with an embedding size of 400, 2 hidden layers, 1000 neurons per layer, dropout 0.2 and batch size 32. We add Gaussian noise of size 0.01 to the embedding layer. A clip norm of 5 is applied, as an extra safety measure against exploding gradients. For each text classification neural network, we add on top of the transferred LM an LSTM layer of size 100 with self-attention and a softmax classification layer. For developing our models, we use PyTorch [105] and Scikit-learn [121].

5.5 Results & Discussion

Baselines and Comparison
Table 5.2 summarizes our results. The top two rows detail the baseline performance of the Bag-of-Words (BoW) and Neural Bag-of-Words (NBoW) models. We observe that when enough data is available (e.g. *Sent17*), baselines provide decent results. Next, the results for the generic classifier initialized from a pretrained LM (P-LM) are shown with and without sequential unfreezing, followed by the results of the proposed model SiATL. SiATL is also directly compared with its close relative ULMFiT (trained on Wiki-103 or Twitter) and the state-of-the-art for each task; ULMFiT also fine-tunes a LM for classification tasks. The proposed SiATL method consistently outperforms the baselines, the P-LM method and ULMFiT in all datasets. Even though we do not perform any elaborate learning rate scheduling and we limit ourselves to pretraining in Twitter, we obtain higher results in two Twitter datasets and three generic.

Auxiliary LM objective
The effect of the auxiliary objective is highlighted in very small datasets, such as *SCv1*, where it results in an impressive boost in performance (7%). We hypothesize that when the classifier is simply initialized with the pretrained LM, it overfits quickly, as the target vocabulary is very limited. The auxiliary LM loss, however, permits refined adjustments to the model and fine-grained adaptation to the target task.

	Irony18	Sent17	SCv2	SCv1	PsychExp
BoW	43.7	61.0	65.1	60.9	25.8
NBoW	45.2	63.0	61.1	51.9	20.3
P-LM	42.7 ± 0.6	61.2 ± 0.7	69.4 ± 0.4	48.5 ± 1.5	38.3 ± 0.3
P-LM + su	41.8 ± 1.2	62.1 ± 0.8	69.9 ± 1.0	48.4 ± 1.7	38.7 ± 1.0
P-LM + aux	45.5 ± 0.9	65.1 ± 0.6	72.6 ± 0.7	55.8 ± 1.0	40.9 ± 0.5
SiATL (P-LM + aux + su)	**47.0** ± 1.1	**66.5** ± 0.2	**75.0** ± 0.7	**56.8** ± 2.0	**45.8** ± 1.6
ULMFiT (Wiki-103)	23.6 ± 1.6	60.5 ± 0.5	**68.7** ± 0.6	**56.6** ± 0.5	21.8 ± 0.3
ULMFiT (Twitter)	**41.6** ± 0.7	**65.6** ± 0.4	67.2 ± 0.9	44.0 ± 0.7	**40.2** ± 1.1
State of the art	53.6	68.5	76.0	69.0	57.0
	[122]	[123]	[124]	[125]	

Table 5.2: Ablation study on various downstream datasets. Average over five runs with standard deviation. *BoW* stands for Bag of Words, *NBoW* for Neural Bag of Words. *P-LM* stands for a classifier initialized with our pretrained LM, *su* for sequential unfreezing and *aux* for the auxiliary LM loss. In all cases, F_1 is employed.

Exponential decay of γ

For the optimal γ interval, we empirically find that exponentially decaying γ to half of its initial value over the number of training epochs provides best results for our classification tasks. A heatmap of γ is depicted in Figure 5.2. We observe that small values of γ should be employed, in order to scale the LM loss in the same order of magnitude as the classification loss over the training period.

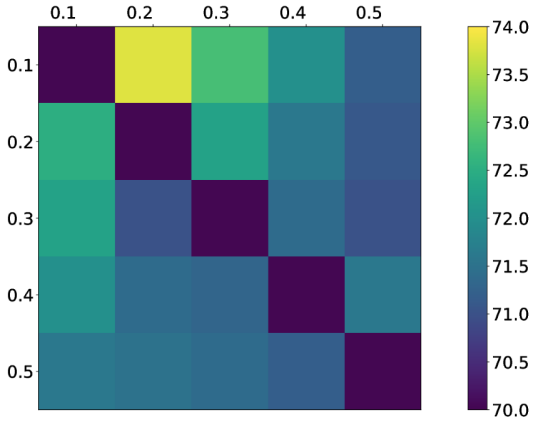


Figure 5.2: Heatmap of the effect of γ to F_1-score, evaluated on *SCv2*. The horizontal axis depicts the initial value of γ and the vertical axis the final value of γ.

Sequential Unfreezing

Results show that sequential unfreezing is crucial to the proposed method, as it allows the pretrained LM to adapt to the target word distribution. The performance improvement is more pronounced when there is a mismatch between the LM and task domains, i.e., the non-Twitter domain tasks. Specifically for the *PsychExp* and *SCv2* datasets, sequentially unfreezing yields significant improvement in F_1 building upon our intuition.

Number of training examples

Transfer learning is particularly useful when limited training data are available. We notice that for our largest dataset *Sent17*, SiATL outperforms ULMFiT only by a small margin when trained on all the training examples available (see Table 5.2), while for the small *SCv2* dataset, SiATL outperforms ULMFiT by a large margin and ranks very close to the state-of-the-art model [124]. As the *Sent17* dataset is by far the largest of the downstream datasets we evaluate our approach on (with approxi-

mately 60K examples), it is not significantly improved by transfer learning. The effect of our SiATL method is highlighted, however, in *SCv2*, which consists of approximately 3K examples. So, our approach is suitable for settings where very limited labeled data are available.

Moreover, the performance of SiATL vs ULMFiT as a function of the training dataset size is shown in Figure 5.3. Note that the proposed model achieves competitive results on less than 1000 training examples for the *Irony18, SCv2, SCv1* and *PsychExp* datasets, demonstrating the robustness of SiATL even when trained on a handful of training examples. We hypothesize that the pretrained part of the model has captured a highly informative word representation. It is important to point out that even if the *SCv2, SCv1* datasets belong to a non-Twitter domain, our model adjusts to different word distributions smoothly and results in a competitive performance when trained on only 500 samples.

Finally, the performance of SiATL in *Sen17* is encouraging. The model obtains a 61% F_1-score on 3000 training samples, or else on less than 10% of the full training set (49K samples).

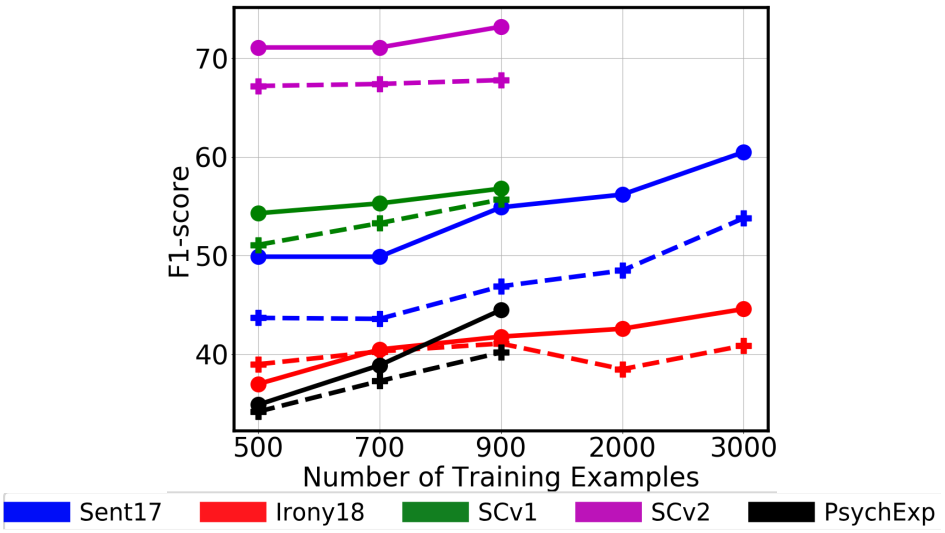


Figure 5.3: Results of our proposed approach (SiATL) (o) and ULMFiT (+) for different datasets as a function of the number of training examples.

5.6 Conclusions & Future Work

We introduce SiATL, a simple and efficient transfer learning method for text classification tasks. Our approach is based on pretraining a LM and transferring its weights to a classifier with a task-specific layer. The model is trained using a task-specific functional with an auxiliary LM loss. SiATL avoids catastrophic forgetting of the language distribution learned by the pretrained LM. Experiments on various text classification tasks yield results competitive to the state-of-the-art, demonstrating the efficacy of our approach. Our method consistently outperforms more sophisticated transfer learning approaches, such as ULMFiT in all tasks.

In the future, we aim to explore further contextual representations from language models. A first step in this direction would be to extend our approach to a bi-directional model, as bi-directional language models have shown to encode deeper information for a given input. We would also like to conduct experiments with different domains. Furthermore, we would like to experiment with subword-level segmentation of our pretraining corpus. That way, the out-of-vocabulary problem can be avoided, which is especially evident in the Twitter domain.

Chapter 6

Conclusions

In this work, we investigate neural transfer learning methods based on deep learning in order to tackle a variety of emotion recognition, sentiment analysis and related classification tasks. Our models allow training deep neural networks to limited training data and obtain a competitive performance. In emotion recognition, transferring a sentiment analysis model to an emotion recognition model can contribute to creating systems capable of analyzing and interpreting human emotions and behaviors. Moreover, pretrained representations captured by language models offer meaningful contextual word representations that encode high-level features of language and capture syntax and semantics. A novel way of using pretrained representations from a language model in a classification model, together with an auxiliary language model objective is proposed. In the context of this thesis, we propose a novel way of leveraging pretrained classification models and the feature representations captured by language models to obtain robust results when there is a scarcity of labeled data.

First, we implement a deep learning model in order to tackle an implicit emotion recognition classification task. We experiment with different transfer learning approaches. Specifically, we use pretrained word embeddings, a pretrained sentiment analysis classification model and pretrained language models. The first two approaches rely on bi-directional multi-layer LSTM models with a self-attention mechanism, while the third one uses a uni-directional multi-layer LSTM model. We use ensembling to determine which approaches are most helpful. By leveraging the representation obtained by a pretrained sentiment analysis model and using it in our emotion recognition classification task, we examine whether a representation learned in a supervised setting for a source dataset can be adapted to a similar dataset efficiently. The idea behind this is that, provided that the source task is more generic than the second task, which is the case in our setting, its feature representation might contain information crucial to improving the performance on the target task and rank close to state-of-the-art.

Second, we propose a novel and conceptually simple transfer learning model that leverages representations from language models and enhances the final transferred classification model with an auxiliary language modeling objective. As a problem that often occurs when fine-tuning a language model to a small labeled dataset is overfitting, which is caused by catastrophic forgetting, we propose a straightforward approach that directly faces this issue. The motivation of adding the auxiliary objective to the transferred model is closely related to multi-task learning. Since our model was pretrained in language modeling, it has learned a feature representation that is generic and models the word distribution of a certain domain. In order to preserve this knowledge, we gradually fine-tune the language model to the classification task at hand. Following this line of thought, we also employ two different training optimizers. The first one is used in the pretrained part of the model and does not permit fast adaptation to the target task, while the second one is used in the newly added layers of the model and results in rapid adjustment to the target task. In order to show the effect of our transfer learning approach, we evaluate on 5 datasets, 3 of which belong to a different domain (namely *Sarcasm Corpus 1*, *Sarcasm Corpus 2* and *PsychExp*) than the pretraining corpus.

Moreover, we can improve our transfer learning model by adding other auxiliary objectives, apart from the language modeling objective. It could be possible that a task that encodes semantics, such as question answering or next sentence prediction would be beneficial as an auxiliary objective. More refined word representations could then be trained, that could be applied to natural language inference and sequence tagging tasks.

A limitation of the proposed approach is that sequential unfreezing requires a careful choice of hyperparameters. In the future, in order to avoid extensive hyperparameter tuning, we plan to model the pretrained language modeling objective and the classification objective as two players involved in a min-max game. Inspired by game theory, we should be able to formulate the problem as adversarial, in a small area around the local loss minimum of the pretrained language model. Then, we could investigate adversarial methods of training. For instance, we could incorporate optimistic Adam [126], that offers the possibility of easing training oscillations caused by Adam optimizer.

Another future direction would be to experiment with domain-adversarial training [127]. The problem of transfer learning is that when source and target distribution are very different, which is often the case, the pretrained model is unable to adjust to the target task. So, an idea would be to build a shared feature representation for different domains, by replacing the auxiliary language modeling objective with a domain classification objective, the target of which would be to predict the domain of a given sentence (source or target). Instead of minimizing this loss, we would intend to maximize it, while also minimizing the target-task classification loss. The intuition behind this approach is that, if the model is unable to distinguish whether a given input belongs to the source or target dataset, it has probably learnt a robust generic feature representation that can tackle both tasks and avoids catastrophic forgetting.

Finally, we plan to incorporate subword information in our language models and experiment with bi-directional architectures. We also plan to pretrain language models on generic corpora, and investigate adaptive schemes for γ decay over training epochs.

Bibliography

[1] S. M. Mohammad, F. Bravo-Marquez, M. Salameh, and S. Kiritchenko, "Semeval-2018 Task 1: Affect in tweets," in *Proceedings of International Workshop on Semantic Evaluation (SemEval-2018)*, New Orleans, LA, USA, 2018.

[2] A. Chronopoulou, A. Margatina, C. Baziotis, and A. Potamianos, "Ntua-slp at iest 2018: Ensemble of neural transfer methods for implicit emotion classification," *arXiv preprint arXiv:1809.00717*, 2018.

[3] A. Chronopoulou, C. Baziotis, and A. Potamianos, "An Embarrassingly Simple Approach for Transfer Learning from Pretrained Language Models," *arXiv e-prints*, p. arXiv:1902.10547, Feb 2019.

[4] A. Karpathy, "Convolutional neural networks for visual recognition." [Online]. Available: http://cs231n.github.io/neural-networks-1/

[5] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, 2010, pp. 242–264.

[6] S. Ruder, "An overview of multi-task learning in deep neural networks," *arXiv preprint arXiv:1706.05098*, 2017.

[7] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[8] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[9] D. Andor, C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov, and M. Collins, "Globally normalized transition-based neural networks," *arXiv preprint arXiv:1603.06042*, 2016.

[10] J. Weston, A. Bordes, S. Chopra, A. M. Rush, B. van Merriënboer, A. Joulin, and T. Mikolov, "Towards ai-complete question answering: A set of prerequisite toy tasks," *arXiv preprint arXiv:1502.05698*, 2015.

[11] K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom, "Teaching machines to read and comprehend," in *Advances in Neural Information Processing Systems*, 2015, pp. 1693–1701.

[12] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[13] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proceedings of the International Conference on Learning Representations*, San Diego, California, 2015.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," 2009.

[15] D. Dong, H. Wu, W. He, D. Yu, and H. Wang, "Multi-task learning for multiple language translation," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, vol. 1, 2015, pp. 1723–1732.

[16] B. Zoph and K. Knight, "Multi-source neural translation," *arXiv preprint arXiv:1601.00710*, 2016.

[17] M. Johnson, M. Schuster, Q. V. Le, M. Krikun, Y. Wu, Z. Chen, N. Thorat, F. Viégas, M. Wattenberg, G. Corrado *et al.*, "Google's multilingual neural machine translation system: Enabling zero-shot translation," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 339–351, 2017.

[18] H. Peng, S. Thomson, and N. A. Smith, "Deep multitask learning for semantic dependency parsing," *arXiv preprint arXiv:1704.06855*, 2017.

[19] K. Zhao and L. Huang, "Joint syntacto-discourse parsing and the syntacto-discourse treebank," *arXiv preprint arXiv:1708.08484*, 2017.

[20] S. Swayamdipta, S. Thomson, C. Dyer, and N. A. Smith, "Frame-semantic parsing with softmax-margin segmental rnns and a syntactic scaffold," *arXiv preprint arXiv:1706.09528*, 2017.

[21] J. Guo, W. Che, H. Wang, and T. Liu, "Exploiting multi-typed treebanks for parsing with deep multi-task learning," *arXiv preprint arXiv:1606.01161*, 2016.

[22] I. Augenstein, S. Ruder, and A. Søgaard, "Multi-task learning of pairwise sequence classification tasks over disparate label spaces," *arXiv preprint arXiv:1802.09913*, 2018.

[23] E. Enguehard, Y. Goldberg, and T. Linzen, "Exploring the syntactic abilities of rnns with multi-task learning," *arXiv preprint arXiv:1706.03542*, 2017.

[24] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[25] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," in *Proceedings of the Annual Meeting of the ACL*, Melbourne, Australia, 2018, pp. 328–339.

[26] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[27] M. Peters, W. Ammar, C. Bhagavatula, and R. Power, "Semi-supervised sequence tagging with bidirectional language models," in *Proceedings of the Annual Meeting of the ACL*, Vancouver, Canada, 2017, pp. 1756–1765. [Online]. Available: http://aclweb.org/anthology/P17-1161

[28] A. M. Dai and Q. V. Le, "Semi-supervised sequence learning," in *Proceedings of the Advances in Neural Information Processing Systems*, 2015, pp. 3079–3087.

[29] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning, "A large annotated corpus for learning natural language inference," *arXiv preprint arXiv:1508.05326*, 2015.

[30] E. F. Sang and F. De Meulder, "Introduction to the conll-2003 shared task: Language-independent named entity recognition," *arXiv preprint cs/0306050*, 2003.

[31] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv:1606.05250*, 2016.

[32] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *Proceedings of the Conference of the NAACL:HLT*, New Orleans, Louisiana, 2018, pp. 2227–2237. [Online]. Available: http://aclweb.org/anthology/N18-1202

[33] G. Qiu, X. He, F. Zhang, Y. Shi, J. Bu, and C. Chen, "Dasa: dissatisfaction-oriented advertising based on sentiment analysis," *Expert Systems with Applications*, vol. 37, no. 9, pp. 6182–6191, 2010.

[34] X. Jin, Y. Li, T. Mah, and J. Tong, "Sensitive webpage classification for content advertising," in *Proceedings of the 1st international workshop on Data mining and audience intelligence for advertising*. ACM, 2007, pp. 28–33.

[35] V. Stoyanov, C. Cardie, and J. Wiebe, "Multi-perspective question answering using the opqa corpus," in *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2005, pp. 923–930.

[36] J. N. Druckman and R. McDermott, "Emotion and the framing of risky choice," *Political Behavior*, vol. 30, no. 3, pp. 297–321, 2008.

[37] R. P. Bagozzi, M. Gopinath, and P. U. Nyer, "The role of emotions in marketing," *Journal of the academy of marketing science*, vol. 27, no. 2, p. 184, 1999.

[38] S. Brave and C. Nass, "Emotion in human-computer interaction," *The human-computer interaction handbook: fundamentals, evolving technologies and emerging applications*, pp. 81–96, 2003.

[39] N. Gupta, M. Gilbert, and G. D. Fabbrizio, "Emotion detection in email customer care," *Computational Intelligence*, vol. 29, no. 3, pp. 489–505, 2013.

[40] S. Voeffray, "Emotion-sensitive human-computer interaction (hci): State of the art-seminar paper," *Emotion Recognition*, pp. 1–4, 2011.

[41] C. Olah, "Visual information theory." [Online]. Available: https://colah.github.io/posts/2015-09-Visual-Information/

[42] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

[43] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2011.

[44] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[45] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[46] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[47] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[48] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[49] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[50] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the International Conference on Learning Representations*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[51] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[52] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[53] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, no. 8, pp. 1735–1780, 1997.

[54] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[55] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[56] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[57] L. Duong, T. Cohn, S. Bird, and P. Cook, "Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, vol. 2, 2015, pp. 845–850.

[58] S. Dhuria, "Natural language processing: An approach to parsing and semantic analysis," *International Journal of New Innovations in Engineering and Technology*, 2015.

[59] C. D. Manning, C. D. Manning, and H. Schütze, *Foundations of statistical natural language processing*. MIT press, 1999.

[60] B. Liu, "Sentiment analysis and opinion mining," *Synthesis lectures on human language technologies*, vol. 5, no. 1, pp. 1–167, 2012.

[61] P. D. Turney, "Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews," in *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, 2002, pp. 417–424.

[62] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up?: sentiment classification using machine learning techniques," in *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*. Association for Computational Linguistics, 2002, pp. 79–86.

[63] J. M. Wiebe, R. F. Bruce, and T. P. O'Hara, "Development and use of a gold-standard data set for subjectivity classifications," in *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*. Association for Computational Linguistics, 1999, pp. 246–253.

[64] Y. Goldberg, "Neural network methods for natural language processing," *Synthesis Lectures on Human Language Technologies*, vol. 10, no. 1, pp. 1–309, 2017.

[65] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[66] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation." in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Doha, Qatar, 2014, pp. 1532–1543.

[67] Z. S. Harris, "Distributional structure," *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.

[68] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning, "Semi-supervised recursive autoencoders for predicting sentiment distributions," in *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics, 2011, pp. 151–161.

[69] X. Wang, Y. Liu, S. Chengjie, B. Wang, and X. Wang, "Predicting polarities of tweets by composing word embeddings with long short-term memory," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, vol. 1, 2015, pp. 1343–1353.

[70] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing lstm language models," *arXiv preprint arXiv:1708.02182*, 2017.

[71] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in neural information processing systems*, 2014, pp. 3320–3328.

[72] L. N. Smith, "Cyclical learning rates for training neural networks," in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2017, pp. 464–472.

[73] S. M. Mohammad and F. Bravo-Marquez, "Wassa-2017 shared task on emotion intensity," *arXiv preprint arXiv:1708.03700*, 2017.

[74] F. Pla and L.-F. Hurtado, "Political tendency identification in twitter using sentiment analysis techniques," in *Proceedings of COLING 2014, the 25th international conference on computational linguistics: Technical Papers*, 2014, pp. 183–192.

[75] A. Tumasjan, T. O. Sprenger, P. G. Sandner, and I. M. Welpe, "Predicting elections with twitter: What 140 characters reveal about political sentiment." *Icwsm*, vol. 10, no. 1, pp. 178–185, 2010.

[76] W. Li and H. Xu, "Text-based emotion classification using emotion cause extraction," *Expert Systems with Applications*, vol. 41, no. 4, pp. 1742–1749, 2014.

[77] J. Si, A. Mukherjee, B. Liu, Q. Li, H. Li, and X. Deng, "Exploiting topic based twitter sentiment for stock prediction," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, vol. 2, 2013, pp. 24–29.

[78] J. Bollen, H. Mao, and X. Zeng, "Twitter mood predicts the stock market," *Journal of computational science*, vol. 2, no. 1, pp. 1–8, 2011.

[79] W. Chamlertwat, P. Bhattarakosol, T. Rungkasiri, and C. Haruechaiyasak, "Discovering consumer insight from twitter via sentiment analysis." *J. UCS*, vol. 18, no. 8, pp. 973–992, 2012.

[80] P. Burnap, W. Colombo, and J. Scourfield, "Machine classification and analysis of suicide-related communication on twitter," in *Proceedings of the 26th ACM conference on hypertext & social media*. ACM, 2015, pp. 75–84.

[81] F. Å. Nielsen, "A new anew: Evaluation of a word list for sentiment analysis in microblogs," *arXiv preprint arXiv:1103.2903*, 2011.

[82] S. M. Mohammad and P. D. Turney, "Emotions evoked by common words and phrases: Using mechanical turk to create an emotion lexicon," in *Proceedings of the NAACL HLT 2010 workshop on computational approaches to analysis and generation of emotion in text*. Association for Computational Linguistics, 2010, pp. 26–34.

[83] A. Go, R. Bhayani, and L. Huang, "Twitter sentiment classification using distant supervision," *CS224N Project Report, Stanford*, vol. 1, no. 12, 2009.

[84] J. Bollen, H. Mao, and A. Pepe, "Modeling public mood and emotion: Twitter sentiment and socio-economic phenomena." *Icwsm*, vol. 11, pp. 450–453, 2011.

[85] S. M. Mohammad, S. Kiritchenko, and X. Zhu, "NRC-Canada: Building the state-of-the-art in sentiment analysis of tweets," *arXiv preprint arXiv:1308.6242*, 2013.

[86] S. Kiritchenko, X. Zhu, and S. M. Mohammad, "Sentiment analysis of short informal texts," *Journal of Artificial Intelligence Research*, vol. 50, pp. 723–762, 2014.

[87] J. Deriu, M. Gonzenbach, F. Uzdilli, A. Lucchi, V. D. Luca, and M. Jaggi, "Swisscheese at semeval-2016 task 4: Sentiment classification using an ensemble of convolutional neural networks with distant supervision," in *Proceedings of the 10th international workshop on semantic evaluation*, no. EPFL-CONF-229234, 2016, pp. 1124–1128.

[88] P. Goel, D. Kulshreshtha, P. Jain, and K. K. Shukla, "Prayas at emoint 2017: An ensemble of deep neural architectures for emotion intensity prediction in tweets," in *Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, 2017, pp. 58–65.

[89] C. Baziotis, N. Pelekis, and C. Doulkeridis, "Datastories at semeval-2017 task 4: Deep lstm with attention for message-level and topic-based sentiment analysis," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, Vancouver, Canada, 2017, pp. 747–754. [Online]. Available: http://aclweb.org/anthology/S17-2126

[90] R. Klinger, O. De Clercq, S. M. Mohammad, and A. Balahur, "Iest: Wassa-2018 implicit emotions shared task," *arXiv preprint arXiv:1809.01083*, 2018.

[91] S. J. Pan, Q. Yang *et al.*, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, no. 10, pp. 1345–1359, 2010.

[92] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014, pp. 1717–1724.

[93] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1701–1708.

[94] A. Agrawal, J. Lu, S. Antol, M. Mitchell, C. L. Zitnick, D. Parikh, and D. Batra, "Vqa: Visual question answering," *Int. J. Comput. Vision*, vol. 123, no. 1, pp. 4–31, May 2017.

[95] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proceedings of the Advances in Neural Information Processing Systems*, 2013, pp. 3111–3119.

[96] S. R. Bowman, C. Potts, and C. D. Manning, "Recursive neural networks can learn logical semantics," *arXiv preprint arXiv:1406.1827*, 2014.

[97] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1631–1642.

[98] R. Nallapati, B. Zhou, C. Gulcehre, B. Xiang *et al.*, "Abstractive text summarization using sequence-to-sequence rnns and beyond," *arXiv preprint arXiv:1602.06023*, 2016.

[99] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, "Bidirectional attention flow for machine comprehension," *arXiv preprint arXiv:1611.01603*, 2016.

[100] J. Howard and S. Ruder, "Fine-tuned language models for text classification," *CoRR*, vol. abs/1801.06146, 2018. [Online]. Available: http://arxiv.org/abs/1801.06146

[101] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Valletta, Malta, May 2010, pp. 45–50, `http://is.muni.cz/publication/ 884893/en`.

[102] S. Rosenthal, N. Farra, and P. Nakov, "Semeval-2017 task 4: Sentiment analysis in twitter," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, Vancouver, Canada, 2017, pp. 502–518.

[103] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[104] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks." *ICML (3)*, vol. 28, pp. 1310–1318, 2013.

[105] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017. [Online]. Available: https://openreview.net/forum?id=BJJsrmfCZ

[106] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and others, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[107] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *arXiv preprint arXiv:1508.07909*, 2015.

[108] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *arXiv preprint arXiv:1607.04606*, 2016.

[109] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, pp. 2493–2537, 2011.

[110] C. Xiong, V. Zhong, and R. Socher, "Dynamic coattention networks for question answering," 2016.

[111] P. Ramachandran, P. Liu, and Q. Le, "Unsupervised pretraining for sequence to sequence learning," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Copenhagen, Denmark, 2017, pp. 383–391. [Online]. Available: http://aclweb.org/anthology/D17-1039

[112] R. Caruana, "Multitask learning: A knowledge-based source of inductive bias," in *Machine Learning: Proceedings of the Tenth International Conference*, 1993, pp. 41–48.

[113] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the International Conference on Machine learning*, 2008, pp. 160–167.

[114] A. Søgaard and Y. Goldberg, "Deep multi-task learning with low level tasks supervised at lower layers," in *Proceedings of the Annual Meeting of the ACL*, Berlin, Germany, 2016, pp. 231–235.

[115] Z. Lin, M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, "A structured self-attentive sentence embedding," *arXiv preprint arXiv:1703.03130*, 2017.

[116] H. G. Wallbott and K. R. Scherer, "How universal and specific is emotional experience? evidence from 27 countries on five continents," *Information (International Social Science Council)*, no. 4, pp. 763–795, 1986. [Online]. Available: https://doi.org/10.1177/053901886025004001

[117] C. Van Hee, E. Lefever, and V. Hoste, "Semeval-2018 task 3: Irony detection in english tweets," in *Proceedings of The 12th International Workshop on Semantic Evaluation (SemEval-2018)*, New Orleans, Louisiana, 2018, pp. 39–50.

[118] S. Oraby, V. Harrison, L. Reed, E. Hernandez, E. Riloff, and M. A. Walker, "Creating and characterizing a diverse corpus of sarcasm in dialogue," in *Proceedings of the SIGDIAL 2016 Conference, The 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 2016, pp. 31–41. [Online]. Available: http://aclweb.org/anthology/W/W16/W16-3604.pdf

[119] S. Lukin and M. Walker, "Really? well. apparently bootstrapping improves the performance of sarcasm and nastiness classifiers for online dialogue," in *Proceedings of the Workshop on Language Analysis in Social Media*, Atlanta, Georgia, June 2013, pp. 30–40. [Online]. Available: http://www.aclweb.org/anthology/W13-1104

[120] E. Loper and S. Bird, "Nltk: The natural language toolkit," in *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, 2002, pp. 63–70. [Online]. Available: https://doi.org/10.3115/1118108.1118117

[121] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, pp. 2825–2830, 2011.

[122] C. Baziotis, A. Nikolaos, P. Papalampidi, A. Kolovou, G. Paraskevopoulos, N. Ellinas, and A. Potamianos, "Ntua-slp at semeval-2018 task 3: Tracking ironic tweets using ensembles of word and character level attentive rnns," in *Proceedings of the 12th International Workshop on Semantic Evaluation (SemEval-2018)*, New Orleans, Louisiana, 2018, pp. 613–621. [Online]. Available: http://aclweb.org/anthology/S18-1100

[123] M. Cliche, "Bb_twtr at semeval-2017 task 4: Twitter sentiment analysis with cnns and lstms," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, Vancouver, Canada, 2017, pp. 573–580. [Online]. Available: http://aclweb.org/anthology/S17-2094

[124] S. Ilic, E. Marrese-Taylor, J. A. Balazs, and Y. Matsuo, "Deep contextualized word representations for detecting sarcasm and irony," in *Proceedings of the 9th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, 2018, pp. 2–7. [Online]. Available: https://aclanthology.info/papers/W18-6202/w18-6202

[125] B. Felbo, A. Mislove, A. Søgaard, I. Rahwan, and S. Lehmann, “Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm,” in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Copenhagen, Denmark, 2017, pp. 1615–1625. [Online]. Available: http://aclweb.org/anthology/D17-1169

[126] C. Daskalakis, A. Ilyas, V. Syrgkanis, and H. Zeng, “Training gans with optimism,” *arXiv preprint arXiv:1711.00141*, 2017.

[127] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, “Domain-adversarial training of neural networks,” *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 2096–2030, 2016.

Appendix A

Abbreviations

(AI): Artificial Intelligence
(ANN): Artificial Neural Network
(Bi-LSTM): Bi-directional LSTM
(BoW): Bag-of-Words
(CBOW): Continuous Bag-Of-Words
(CNN): Convolutional Neural Network
(CV): Computer Vision
(DL): Deep Learning
(DNNs): Deep Neural Networks
(GPU): Graphical Processor Unit
(GD): Gradient Descent
(IE): Information Extraction
(LM): Language Model
(LR): Logistic Regression classifier
(LSTM): Long Short-Term Memory unit
(ML): Machine Learning
(MT): Machine Translation
(MTL): Multi-Task learning
NBoW: Neural Bag-of-Words
(NLP): Natural Language Processing
(POS): Part-Of-Speech
(QA): Question Answering
(RNNs): Recurrent Neural Networks
(SGD): Stochastic Gradient Descent
(SVM): Support Vector Machine classifier
(TL): Transfer Learning