Emotion Recognition on Twitter Using Neural Networks

A dissertation presented

by

Niko Colnerič

to

The Faculty of Computer and Information Science

in partial fulfilment of the requirements for the degree of

Doctor of Science

in

Computer and Information Science

Ljubljana, 2019

APPROVAL

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgement has been made in the text.

— Niko Colnerič —

May 2019

The submission has been approved by

dr. Janez Demšar
Professor of Computer and Information Science
ADVISOR

dr. Igor Mozetič
Senior Researcher
EXTERNAL EXAMINER
Institut Jožef Stefan

dr. Marko Robnik Šikonja
Professor of Computer and Information Science
EXAMINER

dr. Carlo Strapparava
Senior Researcher
EXTERNAL EXAMINER
The Fondazione Bruno Kessler

dr. Marko Bajec
Professor of Computer and Information Science
EXAMINER

PREVIOUS PUBLICATION

I hereby declare that the research reported herein was previously published/submitted for publication in peer reviewed journals or publicly presented at the following occasions:

[1] Colnerič N & Demšar J (2018) Emotion Recognition on Twitter: Comparative Study and Training a Unison Model. *IEEE Transactions on Affective Computing* (in print).

I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as graduate student at the University of Ljubljana.

"We believe that emotions are powerful inner forces that affect our behavior and thoughts, even when we would prefer that they did not."

— Robert Plutchik, *Theories of Emotion*, 1980.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*
Niko Colnerič
Prepoznavanje čustev na Twitterju z uporabo nevronskih mrež

POVZETEK

Čeprav prodira globoko učenje na vsa področja procesiranja naravnega jezika, do zdaj še ni bilo uporabljeno za prepoznavanje čustev. Večina dosedanjih študij prepoznavanja čustev na tvitih uporablja preproste klasifikatorje na značilkah, ki pripadajo modelu vreče besed ali pa jih raziskovalci konstruirajo ročno. Glavna tema disertacije je izboljšava modelov za prepoznavanje čustev v tvitih z uporabo nevronskih mrež. V ta namen najprej ustvarimo tri velike podatkovne množice, sestavljene iz učnih primerov, ki so označeni glede na to, katero čustvo po Ekmanovi, Plutchikovi ali POMS-ovi kategorizaciji izražajo. Čustvene oznake pridelamo avtomatsko z uporabo Twitterjevega mehanizma za samooznačevanje vsebine, s t. i. tematskimi oznakami (angl. *hashtags*). Nato primerjamo natančnost klasifikatorjev z uporabo modelov vreče besed in latentnega semantičnega indeksiranja z natančnostjo nevronskih mrež, tako rekurenčnih kot konvolucijskih, ki na vhodu sprejmejo besede ali znake. Nadalje smo raziskovali prenosljivost reprezentacij končnih skritih stanj modelov nevronskih mrež, natančneje, ali je reprezentacija, naučena pri treniranju modela za neko klasifikacijo čustev, lahko koristna za napovedovanje druge klasifikacije. Zaključimo z učenjem skupnega modela, ki je sposoben prepoznavati čustva vseh treh omenjenih klasifikacij, pri tem pa je omejen na uporabo skupne reprezentacije.

Eksperimentalno pokažemo, da so nevronske mreže natančnejše od klasičnih pristopov k prepoznavanju čustev. Kot najnatančnejše se izkažejo rekurenčne mreže, ki na vhodu sprejemajo znake in tako predstavljajo celosten pristop k učenju (angl. *end-to-end learning*). Čeprav je prenosljivost reprezentacij modelov, ki so trenirani na eni podatkovni množici, precej slaba, se ta drastično izboljša pri skupnem modelu. Pri učenju skupnega modela z znanimi metodami opazimo, da je natančnost zelo neuravnotežena glede na podatkovne množice, predvsem zaradi velike razlike v številu učnih primerov znotraj posamezne množice. Zato zasnujemo novo strategijo treniranja takšnih skupnih

modelov, s katero naučimo model, katerega natančnost je uravnotežena čez vse tri podatkovne množice.

Ključne besede prepoznavanje čustev, tekstovna analiza, Twitter, rekurenčne nevronske mreže, konvolucijske nevronske mreže

University *of Ljubljana*
Faculty *of Computer and Information Science*
Niko Colnerič
Emotion Recognition on Twitter Using Neural Networks

ABSTRACT

Deep learning has recently revolutionised many fields of natural language processing but has not yet been applied to emotion recognition. Most recent studies of emotion recognition on tweets used simple classifiers on a combination of bag-of-words and human-engineered features. Hence, we worked on improving emotion-recognition algorithms using neural networks. To this end, we created three large emotion-labelled data sets corresponding to Ekman's, Plutchik's, and POMS's emotions by exploiting Twitter's popular self-annotation mechanism — hashtags. We compared the performance of bag-of-words and latent semantic indexing models with the performance of neural networks. We trained several word- and character-based, recurrent and convolutional neural networks. Further, we investigated the transferability of final hidden state representations of neural networks: how appropriate is the representation trained on one classification for recognising another one? Finally, we developed a single model for recognising all three emotion classifications from a shared representation.

We show that neural networks can surpass traditional text classification approaches for emotion recognition. Recurrent neural network working directly on characters without any text preprocessing in a completely end-to-end fashion was the most successful architecture. Although models trained on single data sets have revealed poor transferability, we improved the generality of final hidden state representation in the unison model. When training the unison model, the standard training heuristic yielded unbalanced performance, due to the vast difference in data set sizes. However, the newly proposed training strategy produced a unison model with performance comparable to that of single models.

Keywords emotion recognition, text mining, Twitter, recurrent neural networks, convolutional neural networks

ACKNOWLEDGEMENTS

First and foremost I would like to thank my advisor Janez Demšar, for guiding me through the unpredictable waters of academia for the last five years, and Blaž Zupan, for making the Bioinformatics Laboratory an aspiring place to work. I am thankful to the American Slovenian Education Foundation for enabling a research visit to Stanford University, where my research initiated, and to Jure Leskovec and his group for sharing their knowledge. I thank the colleagues from the Bioinformatics Laboratory for all the fun and educational moments we shared. Last but not least, I'm grateful for the support and understanding of my family and friends.

— Niko Colnerič, Ljubljana, May 2019.

CONTENTS

I

Introduction

While the World Wide Web initially consisted of mainly static content prepared by the authors of the site, it soon transformed into a media that anyone can contribute to with ease. The main characteristic of so-called Web 2.0 is that the emphasis is on the user-generated content. Many social networks, blogs, micro-blogging platforms, online encyclopedias, and product reviews sites could not exist without users expressing and sharing their knowledge, experiences, opinions, thoughts, and emotions. While these provide an endless possibility to express oneself, we can not neglect the almost frightening production rate of such content. The manual inspection of web-scale data is usually infeasible, creating the need for automatic systems able of summarising, organising, classifying, and presenting the desired data. Hence, the tools for natural language processing (NLP) and understanding are becoming pivotal in efforts to extract knowledge from the abundance of online content.

The problem of automatic emotion recognition is defined as follows. For a given piece of text, the algorithm should recognise which emotions the author expressed in the writing. As an instance of text categorisation problem, the algorithm should consider a set of pre-defined categories and pick the most suitable for a given text. For example, consider these short texts:

- Someone went into my car during practice yesterday and stole my big hunk out of the console ... I'm pissed!
- I feel so guilty for eating.
- Every time I think about leaving for school at the end of summer I get really bad anxiety and my heart feels like it's breaking.
- Sun in my eyes but I don't mind, what a beautiful day we've had in New York today!
- Do you know that feeling if your mom doesn't allow you to buy the most beautiful dress in the world?
- I love how my dad doesn't notice shit I do, but he notices everything I don't!
- That dream you have that every time you think of it your heart breaks a little because you know it won't come true.

- Not sure if I should look forward to tomorrow or not.
- I need a day off and it's only Tuesday.
- When I write a paper and then it doesn't save!
- My job doesn't stress me out at all.

Some authors, as in the first few examples, are quite transparent about the emotions they want to express (e.g. *"I'm pissed!"*, *"I feel so guilty"*, or *"I get really bad anxiety"*). Others express the emotions more subtly. People overlooking your efforts might cause annoyance, while saying that you need a day off might express fatigue. Both cases are easy for humans to interpret due to the background knowledge we implicitly consider, but they may be trickier for automatic systems. Finally, notice that from texts like *"My job doesn't stress me out at all."* it is almost impossible to infer whether this is a genuine or sarcastic statement. Although some of these examples are quite easy to classify, others are much more difficult — even for humans — which illustrates the challenges we face when developing an automated emotion recognition system.

Applying such algorithms on the web-scale data can be used to gauge public opinions [1], while their utility also extends to predicting real-life events. Observations of online chat activity have been used to predict book sales [2]. Blog posts showed as predictors of product sales performance [3]. There is a correlation between movie's financial success and a context around its references in blog posts [4], and it has been shown that tweets about a movie are a good predictor of its box-office revenues [5]. Multiple studies have used expressions from online content to predict stock market changes [6–8].

Most recent studies tackled the emotion-recognition problem with a typical approach to text categorisation. That is, they first took raw text and transformed it into a fixed length vector representation. These representations usually contain statistics on the number of occurrences of specific words inside the document — known as bag-of-words — accompanied with other, human-engineered features. Next, they passed these representations together with emotion labels to a machine learning algorithm that trained a model for recognising emotions. We refer to these as traditional text classification approaches.

Although these approaches have proven successful, many automatic systems for various NLP tasks recently received a boost in performance when using neural networks [9]. One of the central questions of the thesis is: *can neural networks improve the accuracy of*

emotion recognition systems? As neural networks can recognise more complex patterns than traditional text categorisation approaches, they have the potential to improve emotion recognition systems.

Furthermore, previous studies were mostly recognising only one emotion classification at a time although there is no consensus among psychologists about a universal set of emotions, so multiple such classifications exist. This leads us to the second central question of the thesis: *can we develop a single model for recognising multiple emotion classifications at the same time, with performance comparable to multiple separate models?* As this model can recognise emotions from various classifications at the same time, and since it is trained concurrently on multiple data sets, we refer to it as a *unison model*. Working with multiple classifications enables performance comparisons of different classifications across the same type of data, and tests whether emotions from some classifications are harder to recognise than others. The motivation behind these experimentations is that such multi-task settings can yield better performance due to better generalisation and less over-fitting.

Throughout our experiments we focused primarily on completely autonomous systems in an end-to-end fashion. Hence, human interaction was kept to the minimum or was eliminated entirely from the process of training our models wherever possible.

1.1 Thesis Overview

Chapter 2 presents the background this thesis builds upon. We first introduce three emotion classifications: Ekman's basic set of emotions, Plutchik's wheel of emotions, and Profile of Mood States (POMS). Then we describe approaches to traditional text classification. We next describe neural networks, especially recurrent and convolutional ones, and how they are applied to classification of text documents. We conclude the background chapter with a section on training a single model on multiple data sets, which is the base for building our unison model, and a section on related work.

Chapter 3 focuses on generating training data from a massive data set of tweets. We first describe how we filtered out the tweets that are unsuitable for learning, and then illustrate how hashtags were exploited for creating our target categories.

The experimental setup along with the methodology is the focus of Chapter 4. We first explain experiments with traditional text classification approaches that serve as a baseline for comparison with neural networks. Next, we investigate the transfer abilities

of our neural networks models; more precisely, we test whether the embedding trained on one classification is general enough for predicting a different one. We conclude the chapter with a section about the unison model and present a novel training heuristic.

Experimental results along with discussion about each set of experiments are presented in Chapter 5, while Chapter 6 showcases the utility of our models in Python and Orange. Concluding remarks are provided in Chapter 7.

1.2 Scientific Contributions

- *A novel training heuristic for training neural networks in multi-task settings*

  We proposed a novel training heuristic for training multi-task neural networks. The heuristic focuses especially on settings with data sets of various complexity and data set sizes. The motivation came from observing how researchers get the intuition whether a neural network started overfitting. We used the difference between the accuracy on training and evaluation data sets as a proxy for training progress and exploited it to guide the sampling for the next training batch. We showed that this training approach drastically improved the performance of our unison model, especially for POMS — the largest among our data sets, which underperformed using known training approach.

- *A universal emotion embedding for tweets*

  We developed a unison model for recognising emotions from three emotion classifications at the same time, while sharing the majority of its parameters across tasks. Parameter sharing forced the model to discover general embedding able to recognise emotions from multiple classifications.

1.3 Technical Contribution

- *Publicly available models for emotion recognition*

  Our best performing models are freely available in Python and Orange (see Chapter 6), which enables future studies to compare with our work.

2

Background

We start the chapter by describing three emotion classification schemes we used to classify tweets in Section 2.1 and then focus on the methodology. We describe the traditional approach to text classification with the bag-of-words transformation of texts to vectors in Section 2.2. Using simple classifiers on top of these transformations served as a baseline for comparison. Section 2.3 focuses on neural networks. We present word and character embeddings that served as inputs to networks, recurrent and convolutional neural network architectures, an approach to training such networks, and an example of unified architecture. We conclude the chapter with a review of related work in Section 2.4.

2.1 Emotion Classifications

There is a multitude of discrete emotion classification theories without a consensus on a single one. Hence, we decided to work with the three that are the most popular in natural language processing and have been used in previous studies: Ekman's set of basic emotions, Plutchik's wheel of emotions, and Profile of Mood States (POMS). Working with multiple classifications not only allowed the comparisons between them and with the related work, but it also enabled the development of a single model for recognising all classifications, which can lead to improved performance or better generalisation.

2.1.1 Ekman's Set of Basic Emotions

Paul Ekman is an American psychologist who pioneered the studies of emotions. His work is based on observing facial expressions as a universal way of expressing emotions that cross cultural- end socio-economic borders. In his seminal work entitled *An argument for basic emotions* [10] he defined nine characteristics that distinguish basic emotions. Out of those nine, three can be used to distinguish between different emotions — a challenge we focus on in this thesis — while others can be used to differentiate between emotions and other affective states such as moods, emotional traits, or attitudes. The three characteristics for distinguishing between emotional states are the following:

- *distinctive universal signals*: different facial expressions correspond to different emotions,
- *distinctive physiology*: autonomic nervous system activity differs across basic emotions, and,

- *distinctive universals antecedent events*: as emotions can be a response to fundamental life-tasks, we can distinguish emotions based on the context in which they appear.

Based on these observations he defined a set of basic emotions: *anger*, *disgust*, *fear*, *joy*, *sadness*, and *surprise*. He considers each of these basic emotions not as a single affective state but rather as a family of related states [10].

2.1.2 Plutchik's Wheel of Emotions

Robert Plutchik's *General psychoevolutionary theory of emotion* [11] defined ten postulates, out of which we present four that are the most relevant for our work:

- *Postulate 5*: There is a small number of basic, primary, or prototype emotions.
- *Postulate 6*: All other emotions are mixed or derivative states; that is, they occur as combinations, mixtures, or compounds of primary emotions.
- *Postulate 8*: Primary emotions can be conceptualised in terms of pairs of polar opposites.
- *Postulate 10*: Each emotion can exist in varying degrees of intensity or levels of arousal.

The postulates lead to the development of Plutchik's wheel of emotions depicted in Figure 2.1. It defines eight basic, pairwise contrasting emotions: *joy – sadness*, *trust – disgust*, *fear – anger*, and *surprise – anticipation*. The circular representation reflects the opposition by putting each of the contrasting emotion pair opposite to one another.

For each basic emotion, the wheel also defines different intensity levels. For example, *rage* is an intense expression of anger, and *annoyance* is its milder form.

Beyond the basic emotions, the wheel also presents composed ones. They are depicted between two wings corresponding to basic emotions. For example, *love* is a combination of joy and trust, while *remorse* includes elements of sadness and disgust.

2.1.3 Profile of Mood States

Profile of Mood States [12] is a psychological instrument for assessing the individual's mood state. The test consists of 65 adjectives describing moods or feelings, and a person

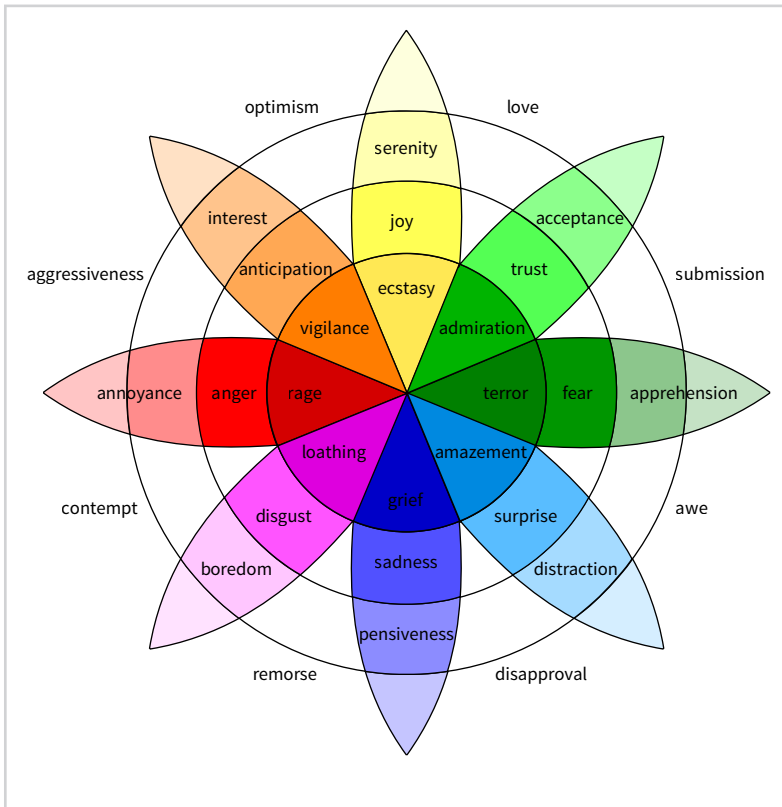


Figure 2.1
Plutchik's wheel of emotions.

is asked to respond on a five-point scale how strongly he experienced a particular mood in the last week. Those answers are aggregated into a seven-dimensional mood state representation consisting of *anger*, *depression*, *fatigue*, *vigour*, *tension*, *confusion*, and *friendliness*. Each adjective contributes to the total score for only one dimension. The greatest majority of adjectives contribute positively to their categories; *not experiencing the mood at all* in the last week contributes zero points to the category while *experiencing it extremely* contributes four. For example, if the person was highly *annoyed* in the last week, this contributes positively to the *anger* category. For some adjectives, however, the rating scheme is inverted — *not at all* contributes four points while *extremely* contributes zero.

The global score for each dimension is obtained by summing the contributions of all adjectives. The higher the score, the more that dimension is expressed in the individual.

The manual for the POMS test [12] along with the instructions on how to perform the test on patients and the normative data is only available to certified psychologists. However, we are only interested in its factor structure, which is publicly available in Norcross et al. [13], and which we further supplemented with the information from the BrianMac Sports Coach website[1]. We emphasise that despite the POMS manual not being public, in our study we only worked with the information that is. We used the following structure (note that adjectives with reverted rating scheme have a "-" sign prepended to the word):

- *Anger*: angry, peeved, grouchy, spiteful, annoyed, resentful, bitter, ready to fight, deceived, furious, bad-tempered, rebellious,
- *Depression*: sorry for things done, unworthy, worthless, guilty, desperate, hopeless, helpless, lonely, terrified, discouraged, gloomy, sad, miserable, blue, unhappy,
- *Fatigue*: fatigued, exhausted, bushed, sluggish, worn out, weary, listless,
- *Vigour*: active, energetic, full of pep, lively, vigorous, cheerful, carefree, alert,
- *Tension*: tense, panicky, anxious, shaky, on edge, uneasy, restless, nervous, -relaxed,
- *Confusion*: forgetful, unable to concentrate, muddled, confused, bewildered, uncertain about things, -efficient,
- *Friendliness*: friendly, clear-headed, considerate, sympathetic, helpful, trusting, and good-natured.

2.2 Traditional Text Classification

Before we can apply a machine learning algorithm to a set of documents — usually referred to as a *corpus* — we need to transform them into a vector representation. A common approach to this is bag-of-words. Given a set of documents, we first construct a set of all words that appear inside any document. Each word will correspond to one

[1] https://www.brianmac.co.uk/pomscoring.htm

dimension in the final vector representation; i.e. the dimensionality of a documents' vector representation is determined by the number of different words appearing in documents. With dimensions defined, a given document is transformed into a vector by counting the number of times a word corresponding to a particular dimension occurs inside a document. An illustrative example of bag-of-words transformation for the following three documents is shown in Table 2.1:

- doc_1: John likes movies. I like movies too.
- doc_2: I like my co-workers.
- doc_3: My successful co-workers make me look incompetent.

Table 2.1
Bag-of-words representation of three documents with column vectors representing each document.

	doc_1	doc_2	doc_3
I	1	1	0
incompetent	0	0	1
John	1	0	0
My	0	0	1
my	0	1	0
co-workers	0	1	1
like	1	1	0
likes	1	0	0
look	0	0	1
make	0	0	1
me	0	0	1
movies	2	0	0
successful	0	0	1
too	1	0	0

Although this looks like a trivial transformation, there are a few critical questions we need to consider. First, what defines a word? How to get a set of words for a given document? A naive approach would split the text by spaces, but that does not work well with

punctuations, dashes, etc. Hence, we usually rely on a pre-trained, language-specific, off-the-shelve tool: a tokeniser. Second, do we want to keep punctuation? In the example above, we did not. However, punctuation might indicate the tone the text is expressing or can influence the meaning, so we might want to keep it. Third, should our analysis be case sensitive or not? Considering the example in Table 2.1 it seems reasonable to group dimensions *My* and *my* into one dimension. Contrary, grouping *us* and *US* would make it impossible to distinguish between the pronoun and the abbreviation for the United States. As this is task-specific, we need to use our best judgement when transforming documents or compare the performance of both approaches. Fourth, do we keep all words or only some? Keeping all words requires more resources when training classifiers and might hurt their performance. Also, it seems useless to keep words that occur in only one document, since they usually bear little predictive power. On the other hand, removing too much might remove some relevant information. A common approach to filtering words is to remove the least and the most common words. Filtering out words from both end of the frequency spectrum should remove words that occur in the majority of the documents as well as those that occur in only a few. Lastly, do we want to group different word variations with same meaning together? For example, *like* and *likes* are just morphological variations of the same verb, so perhaps it is better to group them. Various stemming and lemmatisation algorithms exist that aim to reduce inflected or derived words to their word stem. These can either be a crude set of rules that chop off word endings or methods based on vocabularies and morphological analysis of words.

The aim of the above discussion was not to provide the reader with a set of rules on how to transform documents to vectors, but merely to raise some important questions that need to be considered and to shed light on the required external resources.

Once documents are transformed into vectors, they are, along with the true labels of the documents, fed into a machine learning algorithm, which returns a classifier. For document classification, the following are especially popular: naive Bayes, logistic regression, support vector machines, but others such as decision trees, random forests, k-nearest neighbours can be used as well.

Training a simple classifier on top of the bag-of-words representation of documents is considered a strong baseline for many document classification tasks. However, its primary deficiency, besides a handful of parameters that require tuning, is that it disregards the order of words. Considering the vector representation of doc_3 in Table 2.1

it is impossible to know whether the original sentence was *"My successful co-workers make me look incompetent."* or *"My incompetent co-workers make me look successful."*. Such differences might not be relevant for tasks like topic classification but are of paramount importance in other cases, like emotion recognition, where a more in-depth understanding of the sentence can be required to predict the target category successfully. A usual approach to introducing some context into the bag-of-words transformation are n-grams, which first require setting the value of n and second, they significantly increase the dimensionality of the transformation.

Another flaw of the bag-of-words transformations is its sparseness and high dimensionality. The larger the corpus, the larger the dimensionality of the bag-of-words transformation, especially when working with n-grams. Since for a given document, only some dimensions have non-zero values, various problems, referred to as the curse of dimensionality, occur. A method called Latent Semantic Indexing (LSI) can be used to transform this high-dimensional sparse space into a lower-dimensional dense one. LSI exploits the distributional hypothesis — words with a similar meaning will tend to have similar distributions across large corpora — to shrink the dimensionality of the original space. It does so by grouping the words with similar meaning into one dimension. If the dimensions of original space correspond to words, after using LSI they correspond to concepts. The transformation is done by Singular Value Decomposition (SVD), which transforms the original space into a new orthonormal one. Dimensions are set so that the first one corresponds to the direction of the largest variation in the data. The next one is perpendicular to the first one and is set to capture the most variation left. Hence, when shrinking the dimensionality with LSI, the number of dimensions to keep is usually determined by the proportion of the variance in the data that we want to retain. Studies report that around 70% variance [14] is a reasonable threshold.

LSI shrinks the dimensionality of the space while approximately preserving the distances between documents. Also, the transformation of representation from words to concepts can denoise the original feature space and discover hidden correlations or topics, which can boost the predictive power of classifiers.

2.3 Neural Networks

We present word embeddings, a typical neural network input, in Section 2.3.1. Next, we describe recurrent neural networks in Section 2.3.2 and convolutional ones in Sec-

tion 2.3.3. We conclude by discussing the training of neural networks in Section 2.3.4 and explaining the unison learning approach in Section 2.3.5.

2.3.1 *Word Embeddings*

Word embedding or word vectors are fixed-length vectorial representations of words. Its most straightforward implementation is the so-called one-hot encoding. Similarly to the bag-of-words approach, one-hot encodings have a number of dimensions equal to the number of distinct words in the corpus with each dimension corresponding to one word. Hence, the word vector for word *cat* would have all other dimensions set to zero, except the dimension for a cat which is set to one: $word^{cat} = [1, 0, 0, 0, ..., 0]$.

There are two problems with this embedding that we eluded to in the previous chapter. Such embedding is enormous for a large dictionary, but more importantly, it ignores similarities between words since the distance between any pair of words is exactly the same, regardless if the words are semantically similar or not.

To alleviate these deficiencies, when working with neural networks, we typically use shorter, dense words vectors that resemble semantic similarity between words. Among the most popular ones are *word2vec* and *GloVe*, which were trained in an unsupervised fashion on a large corpus.

Word2vec [15] is a group of two related shallow neural network models: continuous bag-of-words (CBOW) and skip-gram. Both rely on a famous hypothesis by John Rupert Firth: *You shall know a word by the company it keeps*. They were trained on a corpus of Google News by iterating through the training corpus and using small contexts to update the word embedding. A context is a sequence of a few consecutive words. CBOW used all words except the middle one to predict what the centre word might be. Contrary, Skip-gram used the centre word only and tried to predict the context around it. Both models seem to capture the semantic similarity of words well. Since words are now dense vectors, we can find similar words for a given word by finding the word embedding that is the closest to it. For example, the nearest word to the word *car* is *vehicle* and to *My* is *my*, hence the dilemma that we had in Section 2.2 is elegantly resolved by letting word2vec decide which words should be considered equivalent. Further, these vectors are surprisingly accurate in word analogy tasks. For example, a *driver* to a *car* is as a *pilot* to __? To search for an answer with word vectors we employ simple algebraic operations:

$$word^{car} - word^{driver} + word^{pilot} \approx word^{aeroplane}$$

Intuitively, the difference between vectors for a car and a driver is the same as the difference between vectors for an aeroplane and a pilot. The vectors are accurate enough to capture such differences.

Although word2vec was trained by observing only one context at a time, GloVe [16] trained word vectors by considering a global word-word co-occurrence counts as well. GloVe, too, shows the astonishing performance on analogy and word similarity tasks. It provides pre-trained word vectors trained on Wikipedia, Common Crawl, or Twitter.

While the above embeddings did not devote special attention to words with multiple meanings, some of the more recent ones, like ELMo [17], model words characteristics as well as how those vary across different contexts. As this embedding was published while our experimental work was already well under way, we have not included it in our study.

When working on an NLP task with neural networks, it has become a standard practice to start with these pre-trained, off-the-shelve, semantic word vectors. Depending on the task and the amount of training data, these vectors can further be updated when training a network or can remain fixed throughout the training process. Despite these embeddings being usually trained on an enormous corpus, our data set might still contain some words for which embeddings are not provided. For those, we can either use an all-zero vector as the embedding, or use one global embedding for all out-of-vocabulary words, which we randomly initialise and then update during training to learn a single representation for all unknown words.

2.3.2 *Recurrent Neural Networks*

Recurrent neural network (RNN) is a network capable of processing variable-length sequences on input, which makes it particularly suitable for processing text. First, let us define the notation. At time step t we denote the vector for word embedding on input with x_t and the hidden state with h_t. Let n be the length of the input sequence, m the dimensionality of word embeddings on input, d the dimensionality of hidden states, and z the number of outputs of the network. RNN starts with a zero-initialised hidden state h_0. It then processes input vectors one after another and updates the hidden state considering the new input until the input sequence runs out and the final hidden state is produced. Updates are done using dense (fully connected) layers of weights between every neuron on the previous layer and every neuron on the next one. $\mathbf{W}^{(hh)}$ is a matrix of shape $d \times d$ with weights connecting the previous and the current hidden state. $\mathbf{W}^{(hx)}$ is a matrix of shape $d \times m$ with weights connecting the input to the hidden state. At each

time step t the new hidden state is calculated as defined in Equation 2.1, where σ denotes the activation function (e.g. sigmoid, tanh, or ReLu).

$$h_t = \sigma\left(\mathrm{W}^{(hh)} h_{t-1} + \mathrm{W}^{(hx)} x_t\right) \tag{2.1}$$

Notice that the RNN applies the same set of weights, $\mathrm{W}^{(hh)}$ and $\mathrm{W}^{(hx)}$, at every time step. Once the whole input is incorporated into the final hidden state h_n, we use another set of weights $\mathrm{W}^{(s)}$ of shape $z \times d$ that connects the final hidden state to the output. The output of the network, $\hat{y}$, is calculated using an Equation 2.2 for multiclass tasks or Equation 2.3 for multilabel tasks.

$$\hat{y} = softmax\left(\mathrm{W}^{(s)} h_t\right) \tag{2.2}$$

$$\hat{y} = sigmoid\left(\mathrm{W}^{(s)} h_t\right) \tag{2.3}$$

To obtain the output, we apply either softmax or sigmoid activation function where $softmax(x_j) = \frac{e^{x_j}}{\sum_{k=1}^{z} e^{x_k}}$ and $sigmoid(x_j) = \frac{1}{1+e^{-x_j}}$. For multiclass tasks, where each example only belongs to one category, we use softmax as it produces a probability distribution across a set of categories. For multilabel tasks, where each example can belong to multiple categories, the neural network needs to provide a decision for each of them. Hence, we use a sigmoid function that transforms the output for each of the categories to a range [0, 1], which we treat as a probability for a given category. When making predictions in multiclass setting, we pick the category whose probability is maximal. In multilabel settings, we choose those categories whose probability is higher than 0.5. The schematic overview of RNNs is presented in Figure 2.2.

The problem with this vanilla RNN architectures is that although they are theoretically capable of handling long-term dependencies, in practice, they usually perform poorly on data where information from many previous steps is needed. As the hidden state gets multiplied with $\mathrm{W}^{(hh)}$ at every time step, the gradient either explodes or vanishes during training. To alleviate this issue, two improved RNN architectures have been proposed: long short-term memory (LSTM) and gated recurrent units (GRU). Both feature more complicated recurrent steps which we describe in the following sections.

Long Short-Term Memory (LSTM) Similarly to vanilla RNN's hidden state that runs through all stages of the network, LSTM [18] introduces one additional state, called a *cell state*. The cell state serves as a memory, to which LSTM can add or remove information

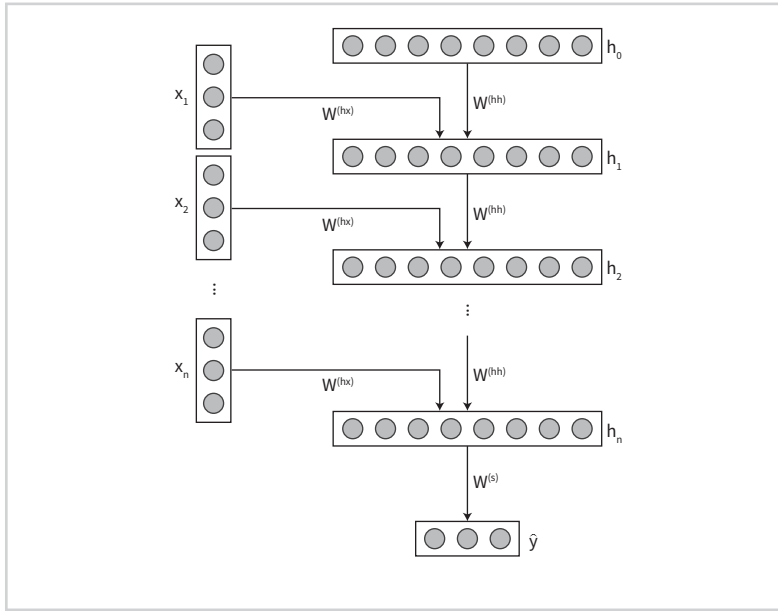


Figure 2.2
The schema of an RNN. We denote the input word embeddings by x_i, the hidden states by h_i, weights between input and hidden states by $\mathrm{W}^{(hx)}$, weights between two consecutive hidden states by $\mathrm{W}^{(hh)}$, weights for the output by $\mathrm{W}^{(s)}$, and the output of the network by $\hat{y}$.

through the mechanism of *gates*. Cell state at time t is denoted as c_t, the outputs of gates as i_t for input gates, f_t for forget gates, o_t for output gates, and the candidate for cell state update as $\widetilde{c_t}$. As before, the dense layer weights are marked with $\mathrm{W}^{(i)}$, $\mathrm{W}^{(f)}$, $\mathrm{W}^{(o)}$, $\mathrm{W}^{(c)}$, $\mathrm{U}^{(i)}$, $\mathrm{U}^{(f)}$, $\mathrm{U}^{(o)}$, and $\mathrm{U}^{(c)}$. The point-wise (Hadamard) product is denoted with $\circ$. At each recurrent stage of LSTM, the cell state and new hidden state are calculated as:

$$f_t = \sigma\left(\mathrm{W}^{(f)}x_t + \mathrm{U}^{(f)}h_{t-1}\right) \tag{2.4}$$

$$i_t = \sigma\left(\mathrm{W}^{(i)}x_t + \mathrm{U}^{(i)}h_{t-1}\right) \tag{2.5}$$

$$o_t = \sigma\left(\mathrm{W}^{(o)}x_t + \mathrm{U}^{(o)}h_{t-1}\right) \tag{2.6}$$

$$\widetilde{c_t} = tanh\left(\mathrm{W}^{(c)}x_t + \mathrm{U}^{(c)}h_{t-1}\right) \tag{2.7}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c_t} \tag{2.8}$$

$$h_t = o_t \circ tanh\left(c_t\right) \tag{2.9}$$

We show the schema of LSTM cells in Figure 2.3.

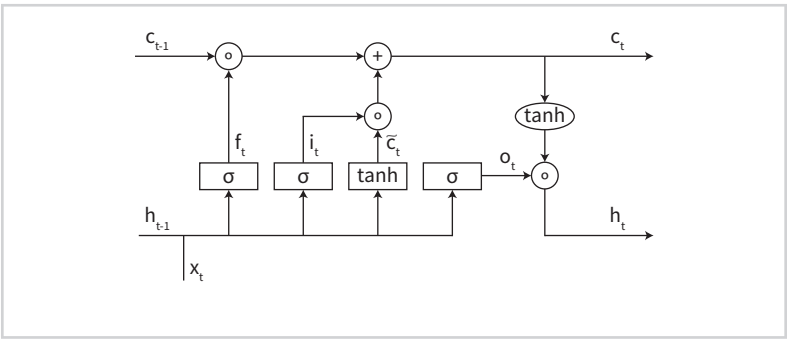


Figure 2.3
LSTM cell. Circles and ovals represent point-wise operations: ∘ for Hadamard product, + for the sum, and *tanh* for the hyperbolic tangent. Rectangles represent neural network layers with different activations: σ for the sigmoid and *tanh* for the hyperbolic tangent. Notice that as all network layers have both h_{t-1} and x_t on input, we only show one line for brevity.

- *Forget gate (equation 2.4)*: a sigmoid layer that decides what information to remove from the cell state c_{t-1}.
- *Input gate (equation 2.5)*: a sigmoid layer that decides what information from the cell update candidate $\widetilde{c_t}$ to store into the new cell state c_t.
- *Output gate (equation 2.6)*: a sigmoid layer that decides what information from the newly updated cell state c_t to send out as the new hidden state h_t.
- *Cell update candidate (equation 2.7)*: a tanh layer that creates a candidate $\widetilde{c_t}$ for updating the cell considering previous hidden state h_{t-1} and current input x_t.
- *New cell state (equation 2.8)*: combines the previous cell state c_{t-1} multiplied by the output of forget gate f_t with the cell update candidate $\widetilde{c_t}$ multiplied by input gate i_t. This stage removes the old information from the cell state and introduces new information according to input and forget gates.
- *New hidden state (equation 2.9)*: considers the output gate o_t and the newly updated cell state c_t to produce the filtered cell state that will serve as a new hidden state.

As gates can block new information from coming into the cell state, it is very easy for information to pass through many time steps almost unchanged, which makes LSTM

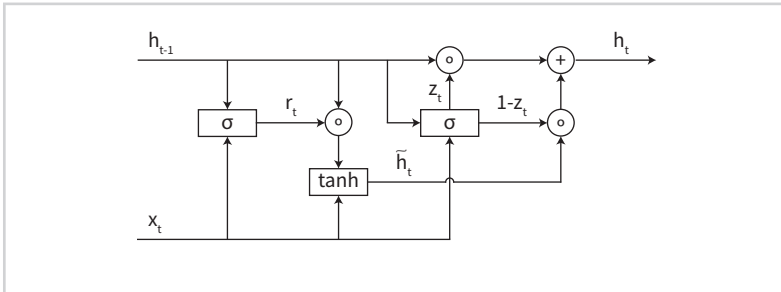


Figure 2.4
GRU cell. Circles represent point-wise operations: ∘ for Hadamard product and + for the sum. Rectangles represent neural network layers with different activations: σ for the sigmoid and *tanh* for the hyperbolic tangent.

much better at modelling long-term dependencies.

Gated Recurrent Units (GRU) GRU [19] is a simplification of LSTM that is also based on the mechanism of gates but does not have a separate cell state. Instead, the gating mechanism is implemented directly on the hidden state. It contains two types of gates: update gate z_t and reset gate r_t. At each recurrent stage of GRU, the new hidden state is calculated as:

$$z_t = \sigma\left(\mathrm{W}^{(z)}x_t + \mathrm{U}^{(z)}h_{t-1}\right) \tag{2.10}$$

$$r_t = \sigma\left(\mathrm{W}^{(r)}x_t + \mathrm{U}^{(r)}h_{t-1}\right) \tag{2.11}$$

$$\widetilde{h_t} = tanh\left(\mathrm{W}^{(h)}x_t + \mathrm{U}^{(h)}\left(r_t \circ h_{t-1}\right)\right) \tag{2.12}$$

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \widetilde{h_t} \tag{2.13}$$

We show the schema of GRU cells in Figure 2.4.

- *Update gate (equation 2.10)*: a sigmoid layer that sets the balance between old hidden state h_{t-1} and hidden state update candidate $\widetilde{h_t}$ that will go into new hidden state h_t.
- *Reset gate (equation 2.11)*: a sigmoid layer that decides how important the previous hidden state h_{t-1} is for generating hidden state update candidate $\widetilde{h_t}$.
- *Hidden state update candidate (equation 2.12)*: a tanh layer that combines the current input x_t with the previous hidden state h_{t-1} updated by the output of reset gate r_t to produce the hidden state update candidate.

- *New hidden state (equation 2.13)*: combines the previous hidden state h_{t-1} and the hidden state update candidate $\widetilde{h}_t$ weighted by the output of update gate z_t.

After the whole input has been incorporated into the final hidden state h_t of either LSTM or GRU networks, the output of the network is obtained with Equations 2.2 and 2.3 as it was for vanilla RNNs. Despite the simplified structure of GRU compared to LSTM, they usually show comparable performance [20].

2.3.3 Convolutional Neural Networks for Text Classification

Although convolutional neural networks (CNN) were designed for computer vision, they were recently shown to be very effective for NLP tasks as well [21–23]. Similarly to the convolution operation applied on small image patches to identify patterns in the image, convolution can be applied to a few consecutive words to extract patterns from written text. On images, convolution usually works across two dimensions, length and height, in a text it is usually applied through one — time. One of the most popular architectures, presented in [21], is shown in Figure 2.5 and will be the topic of this section.

The input to the CNN are word vectors (see Section 2.3.1) stacked horizontally one after the other. Then a set of convolution filters — also known as feature maps — is applied on top of them. In a neural networks context, convolution is a mathematical operation that is given an input matrix x and a set of corresponding weights w, both of shape $n \times m$, and produces a real number activation as defined by Equation 2.14, where $\circ$ stands for point-wise multiplication.

$$convolution(x, w) = \sum_{i=0}^{n} \sum_{j=0}^{m} (x \circ w) \tag{2.14}$$

As each filter is only able to spot one pattern in a given text, we train multiple of them. The required number of feature maps depends on the complexity of the task, with the optimim typically in the range 100-600 [24]. Each feature map has the height equal to the dimensionality of a word embedding and has a variable length or window size. Typically, lengths between 1-10 were shown to be adequate [24]. The size of the window determines the context a feature map can observe at each point. For example, a feature map with a window size of two would scan through the whole input and apply a convolution operation on all pairs of words. Since on each step the convolution window is only shifted for one word, applying a feature map with window size of two on a sequence of 10 words would yield a vector of activations of length 9. For every feature map

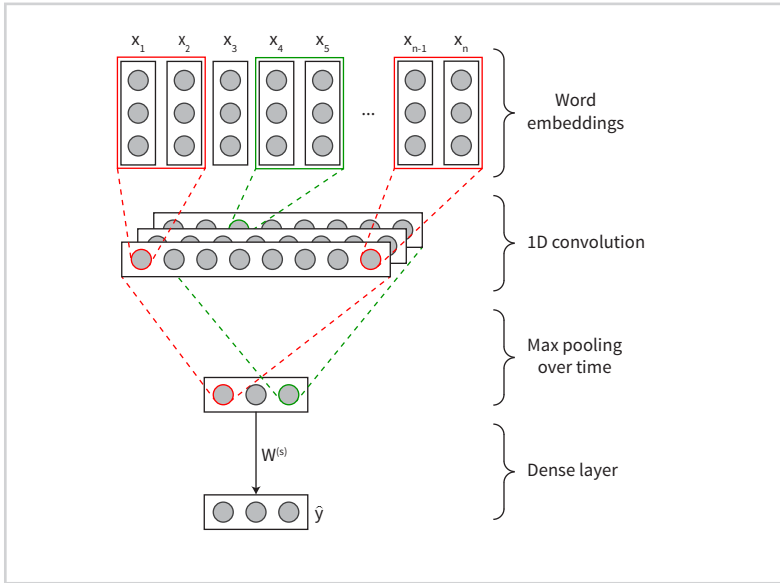


Figure 2.5
The schema of Kim's [21] CNN architecture for text classification. We denote the input word embeddings by x_i, weights for the output by $W^{(s)}$, and the output of the network by $\hat{y}$.

in the network, we get one such vector of activations. These activation vectors are then subjected to a max-over-time pooling as defined by Equation 2.15.

$$max\_pooling(x) = \max_{0 \le t \le n} x_t \tag{2.15}$$

After max-pooling, we have one number for each feature map in the network. It corresponds to the maximal activation of a given feature map anywhere in the text, but without the information about where it occurred. Finally, a dense layer with weights $W^{(s)}$ connects these activations to the output. Again, the output $\hat{y}$ is obtained with the Equation 2.2 or Equation 2.3 for multiclass or multilabel task respectively.

Notice that the above architecture performs convolution on top of words vectors, while some other approaches first decide to quantise texts into images and then treat the problem as an image classification task. Zhang et al. [22] first encoded each character into one-hot representation, then stacked those into a matrix, and treated ones as black pixels and zeros as white to obtain a pictorial representation the input text. Consequently, such approaches call for somehow different architectures, which resemble those used

for classifying images more than the one we describe above for text.

2.3.4 Training of Neural Networks

Training of a neural network is a process that finds such set of parameters with which the neural network performs well for the task at hand. Hence, we first need to define a metric, usually referred to as a cost function, that assesses the quality of a given set of parameters. Let our data consist of training examples denoted by x and their corresponding labels denoted by y. As before, for a given set of parameters w and input x, let the predictions of a neural network be denoted with $\hat{y}$. We define two cost functions, one for multiclass and one for multilabel tasks.

In the multiclass setting, we are training a classifier to predict a probability distribution across a set of mutually-exclusive labels. As each example only belongs to one category, we want a cost function to give a low penalty when the probability of the correct class is high, and a high penalty when the probability of correct class is low. A popular example of such a cost function is *categorical cross-entropy* shown in Equation 2.16.

$$categorical\_cross\_entropy(y, \hat{y}) = -\sum_i y_i \log\left(\hat{y}_i\right) \tag{2.16}$$

In multilabel setting, where examples can belong to multiple categories, we want the classifier to give high probability to all categories to which a given example belongs and a low probability to all others. A popular example of such a cost function is *binary cross-entropy* shown in Equation 2.17.

$$binary\_cross\_entropy(y, \hat{y}) = -\sum_i \left(y_i \log\left(\hat{y}_i\right) + \left(1 - y_i\right) \log\left(1 - \hat{y}_i\right)\right) \tag{2.17}$$

Recall that the output of a neural network $\hat{y}$ is some function f — defined by the architecture of the neural network — of the input x and all neural network parameters w as illustrated in Equation 2.18.

$$\hat{y} = f(x, w) \tag{2.18}$$

Our goal is to find such a set of parameters w that the cost function J will have the minimal possible value. As the function J is differentiable, we can use gradient descent methods to update the set of parameters. We calculate the gradient of a function J with

respect to all parameters w and update them in the direction where the cost function has the lowest value. In its simplest form, the gradient descent algorithm for training a neural network is shown in Algorithm 1.

Algorithm 1 Neural network training by gradient descent.

Input: x, y ▷ data sets
J ▷ a cost function
$\eta > 0$ ▷ learning rate
$\varepsilon > 0$ ▷ stopping criterion
Output: w ▷ a set of neural network weights

1: /* Initialise weights w */
2: **while** $J(x, y, w_{t-1}) - J(x, y, w_t) > \varepsilon$ **do**
3: $w_t = w_{t-1} - \eta \frac{\partial J}{\partial w_{t-1}}$
4: **return** w_t

In the above algorithm, gradients are calculated by considering all examples from our data set. As this gets slow for large data sets, it is more common to calculate the gradients using only a small batch of examples. The algorithm then loops through batches of examples from the data set and uses them to update the weights. We refer to a full training cycle that exhausts all examples from the training set as an *epoch* and several epochs are usually required to train the neural network.

Finally, let us briefly mention that there is a multitude of issues with the above basic gradient descent approach. As the optimisation space of a cost function with regard to neural network weights is highly non-convex, the optimisation might get stuck in local optimum. Also, setting the learning rate η correctly is critical. If it is too large, the training might never converge, and if it is too small, the optimisation will take too long.

There is a myriad of advanced training algorithms that use tricks such as momentum to get through local optimums, and that adapt learning rates throughout the training or with regard to each parameter separately. Some of the most widely-used ones are RMSprop [25], Adagrad [26], Adadelta [27], and Adam [28]. Also, as momentum might cause the cost function J to increase, different stopping mechanisms are employed to stop the training. For example, *early stopping* monitors the performance of the neural network on another set of examples, validation set, and stops the training when the ac-

curacy starts dropping.

2.3.5 *Unison Learning*

So far we focused on training a single neural network to perform a single classification task. Now we investigate the ability of neural networks to perform multiple tasks at the same time. More specifically, given a set of various related tasks, each with its own training data set, how to train a single neural network to perform all given tasks?

The motivation for this comes from the benefits that are usually observed in multi-task settings [29]. First, adding more tasks introduces additional training signals that can improve the performance across tasks in comparison to the performance of models when each task is trained separately. Intuitively, since multiple tasks share a common representation, features constructed for one task might help another. Second, forcing the network to perform various tasks at the same time can steer the training process to prefer more general features that can be useful across tasks. This could lead to less over-fitting and better generalisation of trained models.

Such multi-task learning architectures have one input and multiple outputs — one per task — and rely on *parameter sharing*. All neural network parameters are divided into two groups: common and task-specific ones. Common parameters are those that are shared across tasks, while task-specific ones differ for each task. An example of such architecture that was used by Collobert & Weston [30] is presented in Figure 2.6. After shared word embedding, two task-specific CNN architectures similar to those described in Section 2.3.3 follow.

There are two approaches to train networks that differ in the training data and corresponding labels. In the regular multi-task learning, when training examples are labelled for all tasks, we first calculate the derivatives with respect to all network parameters for each of the tasks separately. Derivatives corresponding to shared network parameters are summed across all tasks, while task-specific derivatives can be used directly. Doing so, we obtain the derivatives for all network parameters and can hence train the network with the optimiser of our choice as described in the previous section.

In the second case, training data is not labelled for all tasks, but each training example is only labelled for one of the tasks. To distinguish this case from the first one, we refer to it as *unison learning*. We define unison learning as an approach to train a multi-task neural network when each training example is only labelled for one of the tasks. Training such a unison network is a bit trickier. As we do not possess the labels for all tasks,

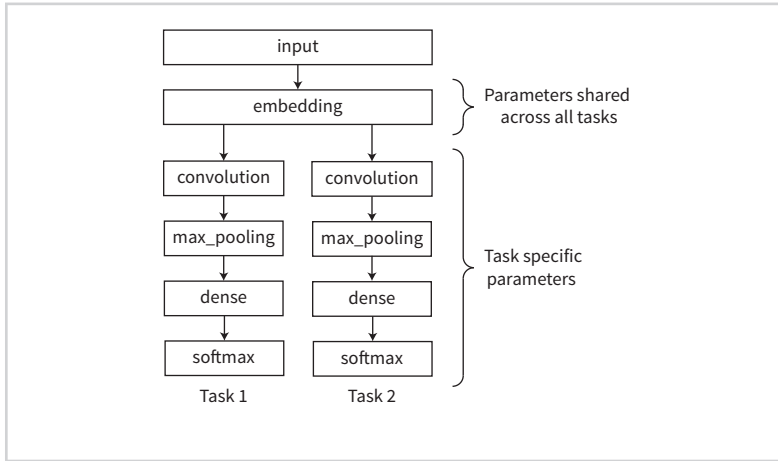


Figure 2.6
An architecture used by Collobert & Weston [30] for multi-task learning with parameter sharing. The word embedding is shared between two tasks, while other NN layers are task specific.

gradients corresponding to missing labels cannot be computed and consequently, we cannot use the gradient summing trick. To train a network in such a setting, Collobert & Weston [30] proposed the following approach that can be summarised as:

1. Select the next task.
2. Select a random training example for this task.
3. Update the NN for this task by taking a gradient step with respect to this example.
4. Go to 1.

Note that in the third step we only update the shared parameters and the task-specific parameters for a current task. Task-specific parameters of the network corresponding to other tasks are left untouched. By iterating through tasks and alternatively updating task-specific parameters, we train a neural network for multiple tasks, without having all training examples labelled for all tasks.

2.4 Related Work

We split the section into multiple paragraphs, each covering a particular research field. First we present recent work on emotion recognition, then we cover sentiment analysis

with a focus on the use of neural networks, and finally we discuss the state of transfer learning.

Emotion Recognition The first recognition models for Ekman's six basic emotions date back at least a decade. Alm et al. [31] annotated each sentence of 185 children's fairy tales but limited their experiments to distinguishing between *emotional* and *non-emotional* sentences and classifying sentences into *no*, *positive*, or *negative* emotion class, without any fine-grained emotion classification. Similarly, Aman & Szpakowicz [32] annotated a corpus of blog posts but again distinguished only between *emotion* and *no emotion* categories. In 2007, SemEval held a competition in emotion recognition from news headlines [33]. However, the main focus was to encourage the study of emotion lexical semantics and consequently no training data was provided. Three out of five competing systems tackled emotion labelling, while others only worked on the polarity classification. The emotion labelling ones were a rule-based system using lexicons [33], a system exploiting Point-wise Mutual Information (PMI) scores gathered through three different search engines [33], and a supervised system using unigrams [33]. Their averaged F1-scores over emotion categories were around 10 %. Performance on this data was later improved to 18 % F1-Score with Latent Semantic Analysis [34]. Chaffar & Inkpen [35] collected a heterogeneous data set of blogs, fairy tales, and news headlines and showed that on this data sequential minimal optimisation SVM yields the greatest improvement over simple baselines. The closest to our approach is the work of Mohammad & Kiritchenko [36] who exploited hashtags corresponding to Ekman's emotion categories to obtain a labelled data sets of 21,051 tweets. With cross-validating SVM on n-grams, they obtained a micro-averaged F1-score of 49.9 %.

The works on Plutchik's emotions include Mohammad & Turney, who created an emotion lexicon using Amazon's Mechanical Turk [37]. Later Mohammad et al. [38] collected a set of about 2,000 tweets concerning the 2012 US presidential election. Besides for emotions, the tweets were also annotated for sentiment, purpose, and style. Using a multitude of custom engineered features like those concerning emoticons, punctuation, elongated words, and negation along with unigrams, bigrams, and emotion lexicons features, the SVM classifier achieved an accuracy of 56.8 %. Tromp & Pechenizkiy [39] developed a rule-based classification technique RBEM-Emo. They trained it on 235 English tweets and achieved 47 % accuracy on a held-out set of 113 tweets.

The works on POMS are rather rare. Johan Bollen led most existing studies. Common

to all is the idea of tracking adjectives defined in the POMS's questionnaire and using its structure to obtain six-dimensional mood representation. Bollen investigated how Twitter mood predicts the stock market changes [6, 40]. In a similar study [41], he correlated emotion time series with records of popular events and showed that such events may have a significant effect on various dimensions of the public mood. By analysing emails submitted to *futureme.org*, Pepe & Bollen revealed the long-term optimism of its users, but medium-term confusion [42]. Those studies used the POMS's questionnaire as a tool for obtaining mood representations but did not study the problem of recognising POMS's categories from the text.

Several studies use other categorisations of emotions. Neviarouskaya and colleagues developed two rule-based systems for recognising nine Izard emotions; the first one works on blogs [43], the second one on personal stories from experience project[2] website [44]. Mishne [45] experimented with detecting 40 different mood states on blog posts from the LiveJournal community. He used features related to n-grams, length, the semantic orientation of words, PMI, emphasised words, and special symbols to train an SVM classifier. Mihalcea & Liu [46] used a subset of these blog posts to train a Naïve Bayes classifier for distinguishing between *happy* and *sad* posts. Yerva et al. [47] fused weather-dependent mood representations from Twitter with real-time meteorological data to provide travel recommendations based on the expected mood of people in a particular city.

Although we approached the problem by essentially predicting hashtags, our study differs from the usual hashtag recommendation [48–50] in that those studies usually choose among tens of thousands of different hashtags with potentially similar meanings, while we target a small set of hashtags corresponding to distinct emotions.

Previous studies showed some promising results for emotion recognition, but none has yet tested the utility of neural networks. Also, we worked with three emotion classifications at the same time to develop a common model able to recognise multiple emotion classifications.

Sentiment Analysis While the use of neural networks for emotion recognition has been limited, they have been widely applied for sentiment analysis. Maas et al. [51] combined the use of supervised and unsupervised techniques to train word embedding that

[2] http://www.experienceproject.com

captures both semantic information and sentiment, and applied it for document-level sentiment polarity task on movie reviews. There have been several studies that applied recursive neural networks to sentiment tasks. Dong et al. [52] used adaptive recursive neural networks to work on target dependant sentiment task; for example, what sentiment is expressed in the tweet towards Google. Socher et al. [53] developed a recursive neural tensor network to compute the sentiment of each node in the parse tree of a sentence. It works in a bottom-up approach and computes the representation of a node by composing those of its children. Doing so, they can capture different scopes of negation, which was illustrated on the newly annotated Sentiment Treebank data set [53]. The data contains sentences along with the corresponding parse trees, which have also been annotated for sentiment at each node of the tree. Tai et al. [54] extended LSTMs to work on parse trees to combine words to phrases by exploiting syntactic properties of natural language. Notice that recursive neural networks require the input data to be of sufficient quality so that parse trees can be produced. Hence, we suspect that such approaches are not the best fit for Twitter, where the used language is often informal and of lower syntactic quality.

Convolutional neural network approaches differ mostly by input representations and consequently architectures. Santos et al. [55] developed a deep CNN that combined character-level as well as word-level embeddings to predict sentiments of tweets, while Severyn et al. [56] described an approach that initialises the weights for deep convolutional networks. Zhang et al. [22] first quantised text to binary images by using a one-hot representation of input characters. As this translated the text classification task to an image recognition one, a standard multi-layer CNN was applied to predict sentiment on multiple reviews data sets. Contrary, Kim [21] applied convolution directly on word embeddings and consequently only one convolutional layer followed by max-pooling sufficed. Kalchbrenner et al. [23] introduced dynamic CNNs and worked directly on sequences of words.

Ghiassi et al. [57] showed that dynamic artificial NN outperforms SVM in the task of determining the consumer sentiment towards a brand. Arkhipenko et al. [58] compared GRU, CNN and SVM for Twitter sentiment detection task at SentiRuEval-2016 and showed that GRU network performed best. Nejat et al. [59] trained NN jointly for both discourse parsing and sentiment analysis using recursive neural network models and showed that such training leads to improvements in both tasks.

The hybrid approach of Wang et al. [60] joined convolutional with a recurrent neural

network to benefit from the long-range features of recurrent as well as coarse-grained local features from convolutional ones. Interestingly, a study by Radford et al. [61] illustrated that sentiment can also be discovered as a side product of training networks for other tasks. While training a recurrent neural network for language modelling, they discovered a neuron whose value directly represents the sentiment of the text on the input. Using the representations learned with their model they achieved excellent results on the binary subset of the Stanford Sentiment Treebank dataset [53]. Furthermore, they showed that by fixing the value of this neuron they could directly influence the sentiment of the generated samples to be either positive or negative.

Transfer Learning Transfer learning is a technique where the knowledge obtained when training the model on one task is re-purposed for another task. It builds on the idea that learning should not start from scratch for each task, but we should instead build on top of the previous knowledge. Lately, transfer learning has become a popular approach for neural networks in multiple fields. For image recognition tasks, it has become a standard practice to employ pre-trained models [62], many of which were trained on ImageNet [63] — an enormous collection of images collected for an object classification competition. Among the most known ones are AlexNet [64] and Inception [65]. It has been shown that first layers of deep CNN tend to discover general features similar to Gabor filters [66] while the higher layers encode more and more specific features. Hence, it is natural to exploit the knowledge from these pre-trained model rather than starting the training from scratch for each of the tasks.

Similar trends have also been observed in natural language processing. We already described one such case in the previous section, where Radford et al. [61] used the neuron trained on language modelling task for predicting sentiment. Otherwise, the most evident example of transfer learning is the use of pre-trained word embeddings. These word embeddings, which are usually trained on a large corpus in an unsupervised manner, contain a semantic representation of words, which we might not be able to learn from scratch on a smaller target data set. Some of the most popular word embeddings are word2vec [15], GloVe [16], and ELMo [17]. See Section 2.3.1 for a more detailed discussion of word embeddings.

3

Data

The data set that we used in this study contains tweets that have been continuously collected through the Twitter API from August 2008 till May 2015. Spanning over seven years, the collection contains roughly 73 billion tweets that occupy about 17 TB of disk space in uncompressed text files. All tweets were imported into a Hadoop cluster running on 40 nodes using 160 hard drives. To allow for quick scans through the data set, we developed a custom map-reduce search application that can scan through the data in approximately one hour and a half.

Since we tackled emotion recognition as a supervised machine learning task, we first needed to construct a data set labelled with emotions on which the classifiers were trained. A common approach to this is to employ a set of human annotators that read the content and pick the most suitable category. For the scale of our data, such a manual approach is too time- and resource-consuming. Also, as the examples from Chapter 1 demonstrate, labelling short texts for emotions is non trivial and ambiguous. Hence we opted for an automated process. Luckily, Twitter provides a mechanism for self-annotation of the content that is quite popular — hashtags. A hashtag is a concatenation of a # sign with an arbitrary word or multi-word phrase, and its primary purpose is to enable efficient search for tweets. As such, the users frequently add hashtags that sum up the content of the tweet. These can be considered as the author's annotation of their content and hence can be exploited for creating labelled data sets. Such an approach, where labels are not assigned by human annotators but come from the data itself, is referred to as *distant supervision*.

We are aware of the danger that such data might be of a lower quality than humanly annotated data sets. We still chose to use this approach because it enables us to use a much larger data set. Deep neural networks require a lot of data and it is a common practice to trade the quality of the data for its quantity. Moreover, in this particular case we suspect that the agreement between human annotators would often be low and hence the use of distant supervision should not significantly decease the quality of the data.

The idea of using hashtags for creating labelled data sets was already proven successful in many recent studies. It has been used in sentiment classification [1, 67, 68], sarcasm detection [69, 70], and personality traits studies [71]. Similarly to our approach, Mohammad et al. [36] created an emotion labelled data set by extracting hashtags.

3.1 Labelling by Distant Supervision

To label the data with emotion categories, we searched among English tweets for exact matches of hashtags corresponding to emotions. For Ekman's set of basic emotions, we searched for *#anger*, *#disgust*, *#fear*, *#joy*, *#sadness*, and *#surprise*. For Plutchik's set of eight emotions, we searched for *#anger*, *#disgust*, *#fear*, *#joy*, *#sadness*, *#surprise*, *#trust*, and *#anticipation*. Although Plutchik defined a more detailed representation, we decided to use only eight main categories and discarded different intensity levels. Also, notice that Ekman's categories are a subset of Plutchik's, which was taken into account when making comparisons. For POMS we discarded the seventh category *friendliness* as it is considered to too weak for valid scoring [13] and only used adjectives for the first six. Among them, we removed the ones that have a negative contribution to their corresponding category (i.e. *relaxed* and *efficient*) and we removed the adjective *blue* as it is mostly used to describe colour, not emotion. We truncated multi-word phrases to a single word without spaces. The final set of hashtags we searched for is the following: *#angry*, *#peeved*, *#grouchy*, *#spiteful*, *#annoyed*, *#resentful*, *#bitter*, *#readytofight*, *#deceived*, *#furious*, *#badtempered*, *#rebellious*, *#sorryforthingsdone*, *#unworthy*, *#worthless*, *#guilty*, *#desperate*, *#hopeless*, *#helpless*, *#lonely*, *#terrified*, *#discouraged*, *#gloomy*, *#sad*, *#miserable*, *#unhappy*, *#fatigued*, *#exhausted*, *#bushed*, *#sluggish*, *#wornout*, *#weary*, *#listless*, *#active*, *#energetic*, *#fullofpep*, *#lively*, *#vigorous*, *#cheerful*, *#carefree*, *#alert*, *#tense*, *#panicky*, *#anxious*, *#shaky*, *#onedge*, *#uneasy*, *#restless*, *#nervous*, *#forgetful*, *#unabletoconcentrate*, *#muddled*, *#confused*, *#bewildered*, and *#uncertainaboutthings*.

For Ekman and Plutchik we used the hashtag itself as the target category, i.e. a tweet containing hashtag #anger is considered as a training example for anger category. For POMS, the hashtags were only considered as proxies to the corresponding category as defined by the POMS structure. That is, a tweet containing #angry was considered as a training example for anger category and so were tweets containing #furious or any of the other 10 adjectives that belong to anger category.

Although the vast majority of tweets contain only one emotion hashtag, there are some that contain multiple of them. For Ekman there are 0.62 % of such tweets, for Plutchik 0.61 % and for POMS 1.16 %. To account for them properly, we conducted our experiments in two settings: multiclass and multilabel. In the multiclass setting, we limited each tweet to express only a single emotion. We took the first emotion hashtag that was found in the tweet, set it as a target category, and discarded any other that might

Table 3.1

Examples of tweets along with the correct target values our classifiers are trained to recognise. In the multiclass mode, the first emotional hashtag is set as the target. In the multilabel mode, for each emotion category, classifiers have to state whether it is expressed (✓) or not (×).

Tweet content	Multiclass	Multilabel					
		anger	disgust	fear	joy	sadness	surprise
I'm so happy to be part of this family. Love to you all! #home #joy #love	joy	×	×	×	✓	×	×
Seeing my twin cycle ... hurts a lot ... #anger #sadness	anger	✓	×	×	×	✓	×
The mix of emotions running through my body right now are ridiculous.. #disgust #Sadness #fear	disgust	×	✓	✓	×	✓	×

come later in the content. This decision is based on the assumption that the user first wrote the most important emotions and that the ones he used later are less relevant. The classifiers in this mode know that exactly one of the target category is correct for each tweet and only have to pick the most suitable one.

In the multilabel setting, each tweet can belong to multiple categories, potentially even to all. We used all emotion hashtags found in the tweet and set them as true categories. The classifiers in this setting have to decide for each emotion category whether it is expressed in the tweet or not, regardless of the decisions for other categories. Not only that multilabel setting accounts for all emotional hashtags that the user provided, but it is also closer to a real-life scenario in which tweets can be non-emotional as it supports a scenario in which all classifiers return a negative prediction. Some examples of tweets, along with their target categories in multiclass or multilabel settings are shown in Table 3.1.

This resulted in a collection of tweets as well as their emotional categories. However, some of them are not suitable for learning as hashtags are not always used for annotating

the content of the tweet. In some cases, the hashtag is used as a crucial part of the content, as people might prepend a # character to any word inside a sentence. Other tweets contain only hashtags, @mentions, links, emoticons, and other non-content bearing words. Some examples of such tweets are shown in Figure 3.1. Since our goal is to develop a model that can recognise emotions from the *textual content* — and not merely find correlations between certain @mentions or links and emotions or recommend one hashtag from the presence of the other — such tweets are undesirable in the training data. We removed them from our training set using two filtering approaches described below.

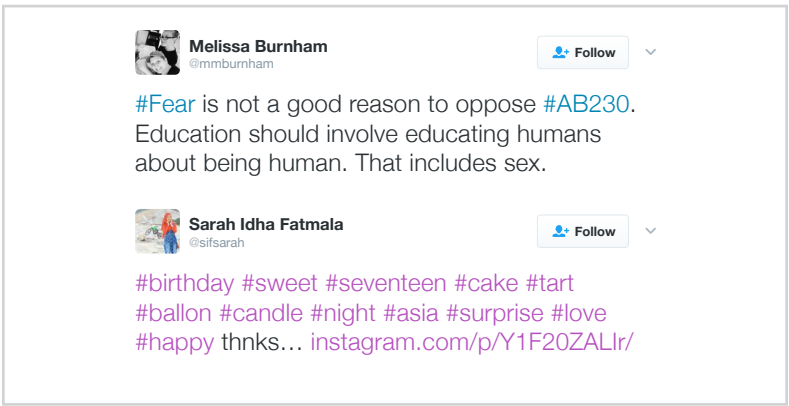


Figure 3.1
Example of tweets not suitable for learning. In the first tweet hashtag *#Fear* is used as a crucial part of the content and not as a label of the content. The second example contains too few content words for recognising emotions. Such tweets were removed from the training set.

To detect tweets without content, we defined the concept of a *content word*. We took the tweet and passed it through a part-of-speech tagger that was designed for Twitter [72]. We defined the content word as any token inside the tweet that was *not* tagged as Twitter- or online-specific (e.g. hashtag, @mention, URL) or Miscellaneous (e.g. number, punctuation). Next, we calculated the content fraction c as shown in Equation 3.1.

$$c = \frac{number\ of\ content\ words}{number\ of\ all\ words} \tag{3.1}$$

After observing the distributions of content fraction across all three data sets, shown in Figure 3.3, and checking some examples, we set the threshold to 0.5. That is, all tweets with c less than 0.5 were removed from our data sets.

To remove tweets where hashtags are used as a crucial part of the content and not as an annotation, we considered the position of the hashtag. We built on a simple idea

Figure 3.2
Examples of tweets that passed all filtering requirements. Note that emotional hastags are used as targets and were removed from the contents of the tweets.

that hashtags that appear towards the end of the tweet are most likely annotations, while ones that appear before might be a part of a sentence. We formalised the concept of a hashtag depth d as proposed in Equation 3.2.

$$d = \frac{number\ of\ content\ words\ before\ hashtag}{number\ of\ content\ words} \tag{3.2}$$

The distribution of hashtag depths across all data sets is shown in Figure 3.4. Again, after inspecting the data and distributions, we set the threshold to 0.9; i.e. all tweets with d less than 0.9 were filtered out since the hashtags might be a crucial part of the sentence and not a label. Gonzales-Ibanez et al. [69] applied a similar procedure but required that the hashtag appeared at the very end of the tweet, which seems too strict. For example, we allow another hashtag or emoticon after the emotional hashtag.

Finally, re-tweets and duplicates were removed from the data set to prevent any training example from leaking into the test set. Also, emotional hashtags, that were used for setting the target categories were removed from the content, lest the models would only learn to spot keywords in the text. Examples of tweets meeting all above requirements are shown in Figure 3.2.

3.2 Data Set Statistics

For each emotion classification, all tweets meeting the requirements were randomly split into three sets: train (60 % of tweets), validation (20 % of tweets), and test set (20 % of tweets). The train set was used for training classifiers while the validation set served for parameter selection. Once all models' parameters were set, we used them for training a classifier on the combination of the train and validation set and assessed their performance only once on the test set.

As the Ekman's categories are a subset of Plutchik's, the split was done simultaneously: the Ekman's train, validation, and test sets are subsets of Plutchik's rather than being sampled independently. This assures that no tweet from the Ekman's train set can be present in the Plutchik's test set and vice-versa. This requirement ensures a fair evaluation of our models' transfer capabilities (see Section 4.3).

The numbers of tweets in each set along with the statistics on the number of discarded tweets are shown in Table 3.2 while the distributions of emotion categories are shown in Figure 3.5.

Table 3.2

Data set sizes. *All* corresponds to all English tweets containing any emotional hashtag, and *filtered* shows the remaining number of tweets after depth, content, and duplicates filtering. These sets were further split into three subsets: *train*, *validation*, and *test*.

	Ekman	Plutchik	POMS
All	1,175,847	1,740,750	9,592,460
Filtered	535,788	798,389	6,536,280
Train	321,461	479,033	3,921,768
Validation	107,183	159,678	1,307,256
Test	107,144	159,678	1,307,256

We observed that class distributions are quite imbalanced. One reason for this, apart from the people's tendencies to express particular kinds of emotions on Twitter, may be the restriction to exact matches of hashtags. While #disgust is not a popular hashtag, its variations (e.g. #disgusted or #revulsion) might be more popular. However, we decided for the exact matches to directly follow emotions as defined by psychologists, which already yielded sufficiently large data sets. By not including word derivatives the

resulting data sets might be more coherent and we did not need to subjectively decide how far to expand the set of hashtags. For example, are *#delight*, *#pleasure*, *#euphoria*, and *#ecstasy*, while being the synonyms of joy, semantically close enough so we could have used those as well when searching examples for the joy category? We leave the idea of expanding these with synonyms for future work.

In comparison with previous studies, our data sets have been collected over a longer period of time; hence they are less influenced by temporal variations like popular or tragic events. They are also orders of magnitude larger than what has been used in previous studies [36] of emotion recognition on Twitter data.

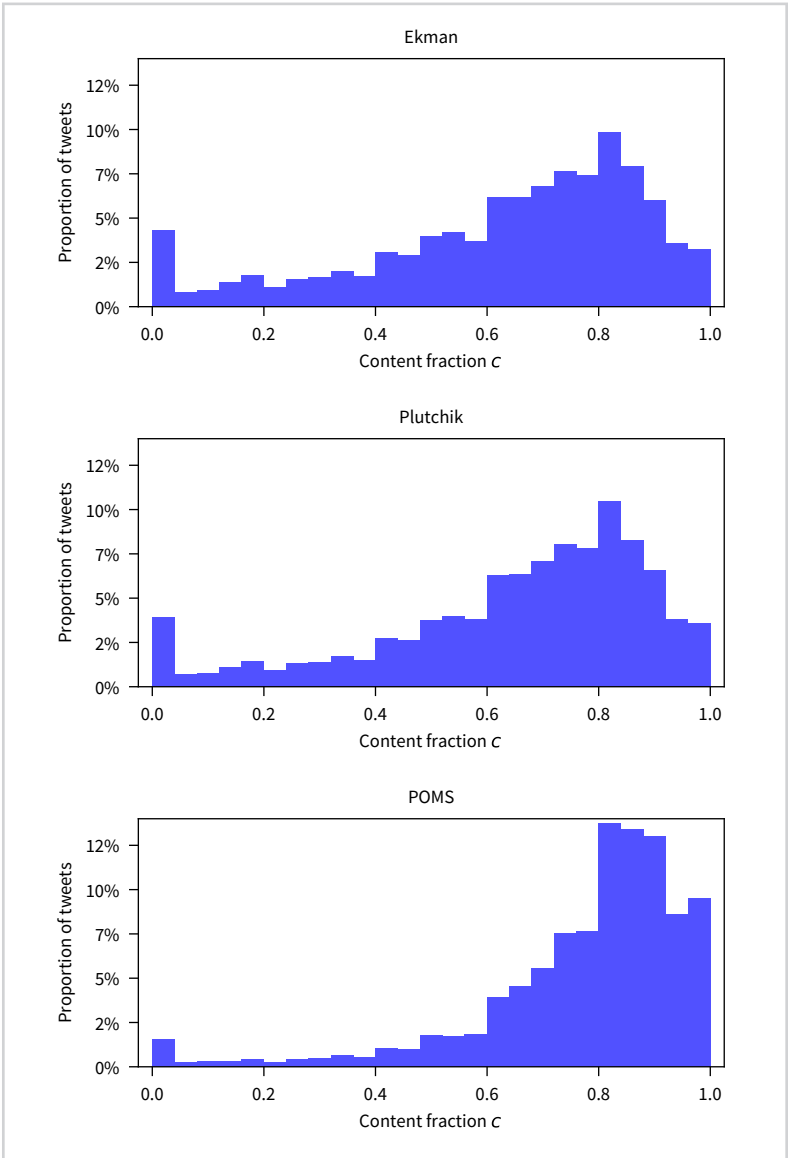


Figure 3.3
The distribution of content fraction c before filtering tweets. Notice that a vast majority of tweets has a content fraction larger than 0.5.

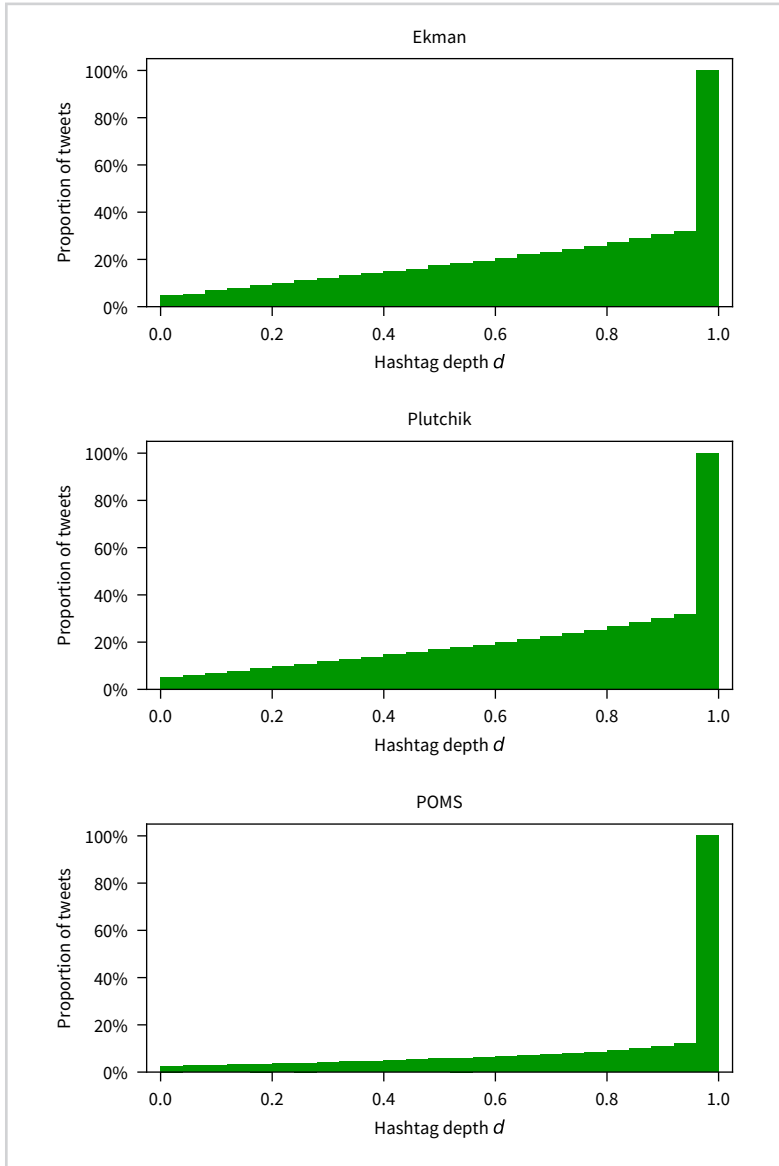


Figure 3.4
The cumulative distribution of hashtag depth d before filtering tweets. Notice that a vast majority of hashtags has a depth of one — i.e. they appear at the very end.

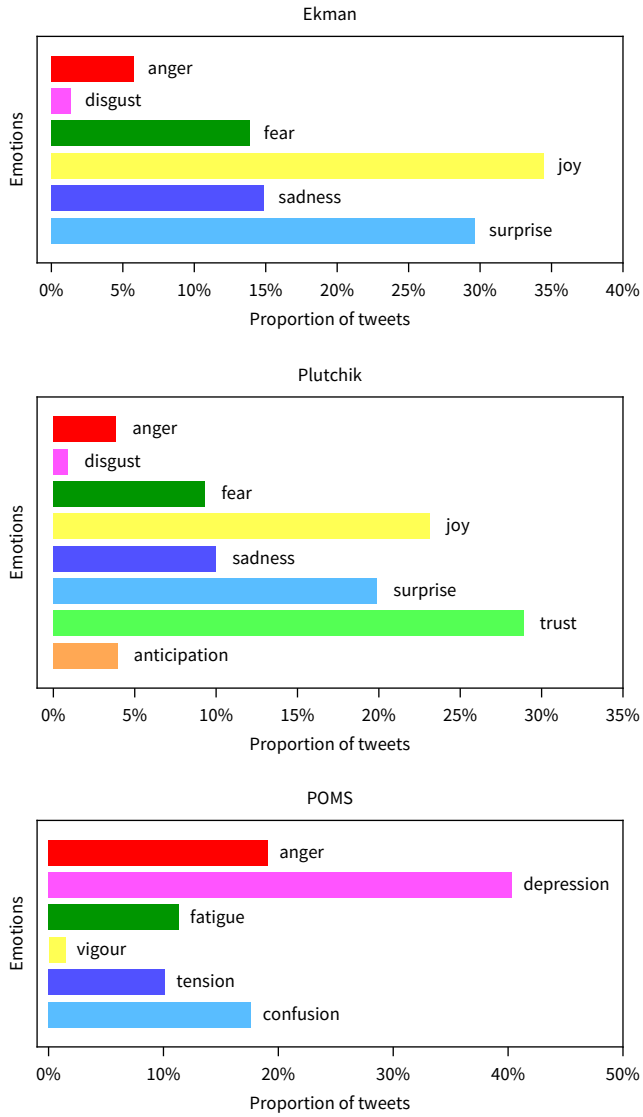


Figure 3.5
Class distributions for the whole data sets. The distributions of *train*, *dev*, and *test* subsets show only negligible differences.

4

Methods

This chapter presents the methodology used in our experimental setup. To tackle our first question, whether neural networks can improve the accuracy of emotion recognition systems, we first describe the set of experiments that set the baselines for comparison with neural networks. Section 4.1 describes the transformations applied to raw text into bag-of-words and the latent semantic indexing models on top of which simple classifiers were applied. With the baselines set, we turn our attention to neural networks. Besides a more popular approach of using words as an input to the neural network, we also experimented with an end-to-end learning approach and tested how networks perform directly on a stream of unprocessed characters. Hence, in Section 4.2 we first describe the word and character embeddings that served as the input and then continue with a description of recurrent and convolutional network architectures.

To test the generality of the trained neural network models, we designed a set of transfer learning experiments, which are the focus of Section 4.3. We investigated whether our neural networks had discovered features that are useful for recognising other emotion classifications as well, not just the one they have been trained for.

Finally, the Section 4.4 describes experiments regarding our second guiding question: whether we can train one neural network capable of predicting all three emotion classifications. We start with the training strategy presented by Collobert & Weston [30] and propose two novel training heuristics that we used to train our unison models.

All experiments were conducted in the Python programming language and we cite the major dependencies used in each of the following sections.

4.1 Traditional Text Classification

4.1.1 Bag of Words

We split the raw text of the tweet into tokens by using the Tweet POS tagger [72], which was designed to handle Twitter's messy language and its specifics such as @mentions, and hashtags. To filter out very infrequent words, which usually have too little predictive power, we removed all tokens that occurred in less than five tweets. As the language on Twitter is very informal and contains many invalid or elongated words, references to other users and URLs, we experimented with two token normalisation techniques before we constructed the bag-of-words representation. *Vanilla BoW* is a model without any normalisation of tokens. *Normalised BoW* grouped some tokens into the same

token by applying these transformations:

- all *@mentions* were truncated to a single token <*user*>,
- all *URLs* were truncated to a single token <*url*>,
- all *numbers* were truncated to a single token <*number*>,
- three or more same consecutive characters were truncated to a single character (e.g. looooooove → love),
- all tokens were lower-cased.

The idea behind these normalisations is that some tokens are too specific and that grouping them into the same token might help the performance of classifiers. For example, instead of having a bunch of separate tokens for *looove*, *loooove*, *looooove* etc. the normalised BoW truncated them into a single token *love*. These transformations drastically shrunk the dimensionality of the feature space which also helped to reduce the over-fitting problems. We compared the performances of both approaches to determine whether these normalisations removed noise from the data and helped the classifiers, or whether they removed meaningful signal and consequently hurt them.

To give the classifiers a bit more context than just single words, we also experimented with the combination of *unigrams* and *bigrams*. For each of the two token normalisation techniques, we created two bag-of-words representations: one containing only unigrams and one containing both unigrams and bigrams. From here on, we refer to the combination of unigrams and bigrams simply as bigrams. For each of the four different bag-of-words transformations we left out all tokens that occur inside the corpus less than five times. The dimensionality of feature spaces for all bag-of-words models is shown in Table 4.1.

4.1.2 Latent Semantic Indexing

As the dimensionality of bag-of-words is extremely high, we also experimented with dimensionality reduction technique that is commonly used for text: latent semantic indexing (LSI). We started with four bag-of-words representations presented in the previous section and applied LSI to each of them. We determined the number of dimensions needed to retain 70% of the variance in the data. While the threshold comes from [73],

Table 4.1
The number of features of BoW and LSI models for combined train and validation sets using different token normalisations. The name *bigrams* stands for a model consisting of a combination of unigrams and bigrams.

	Ekman		Plutchik		POMS
	BoW	LSI	BoW	LSI	BoW
Unigrams Vanilla	45,484	523	58,146	500	183,727
Unigrams Normalised	35,555	316	44,009	299	129,841
Bigrams Vanilla	204,453	5,433	284,467	6,183	1,248,037
Bigrams Normalised	187,533	3,955	256,889	4,390	1,081,598

the number of retained dimensions, which is reported in Table 4.1, is in the range that empirical studies showed as appropriate [14]. LSI experiments were only performed for Ekman and Plutchik, since calculating the SVD decomposition for POMS showed infeasible with the computation resources at our disposal. Both bag-of-word and LSI transformations were performed with Gensim [74].

4.1.3 Classifiers

We selected four classifiers whose performance was tested on bag-of-word and latent semantic indexing representations: support vector machines with linear kernel (SVM), naïve Bayes (NB), logistic regression (LogReg), and random forests (RF). For each of them, these are the parameters we optimised:

- SVM: the regularisation parameter C from 0.001 – 3.0
- NB: no parameters,
- LogReg: the regularisation parameter C from 0.001 – 3.0
- RF: the number of trees from 50–5000; the number of features from 1–600.

Notice that we performed a reduced parameter search for RF on POMS, because training was extremely slow. For example, training 200 trees using bigram vanilla representation took about three days on 40 parallel cores. Hence, for POMS we only experimented with up to 200 trees and left the number of features at its default value (i.e. $\sqrt{n}$, where n is the number of features). We used the classifiers from the scikit-learn [75]

library and left all other parameters[1] at their default values. For a detailed description refer to the documentation at http://scikit-learn.org/stable/documentation.html.

4.2 Neural Network Models

All neural network experiments were conducted at two input granularity levels. First, similarly to traditional approaches, we tokenised the tweet and used words as input. Here the task of the neural network is to read words one by one and learn how to combine them into a suitable representation for recognising emotions. Second, in an end-to-end learning fashion, we trained neural networks directly on the raw text content — the input to the neural network was an unprocessed stream of characters from the tweet. The neural networks in this setting have a more challenging task; they have to learn to combine characters into a meaningful representation for emotion recognition. If in the first setting the concept of words was already encapsulated in the neural network input, the networks in the second task have to first learn words themselves — since space character was not treated any differently than any other character — and then use them to obtain suitable representations.

There are several advantages to the character-based approach. First, we do not need a tokeniser as there is no need to split the raw text into tokens. Secondly, word normalisation is not required. In the word-based approach, we need to decide which morphological variations of words are similar enough that the same token can be used for their representation. For example, should the neural network distinguish between *play* and *playing*? If it does, we need to have different input representations (i.e. embeddings) and if not we can have only one for both of them. All those decisions were left to the neural network to figure out during the learning process. Since our character-based approach does not need any language-dependent tools or any human intervention, we consider it a true end-to-end learning approach. As such, it can easily be tested on other data set and languages.

[1] *Loss*, *penalty*, *tol*, and *max_iter* for SVM; *alpha* for NB; *max_iter*, *solver*, and *tol* for LogReg; *criterion*, *max_depth*, *min_samples_split*, *min_samples_split*, *min_samples_leaf*, *min_weight_fraction_leaf*, *max_leaf_nodes*, *min_impurity_split*, *bootstrap*, and *oob_score* for RF.

4.2.1 Embeddings

For our word-based experiments, we used GloVe [16] embedding as it is the only one we know of that was trained on Twitter data. Hence, our tokenisation and token normalisation followed their approach so the percentage of tokens contained in the dictionary was as high as possible. We used the same normalisation as for *normalised bag-of-words* (see Section 4.1.1) with one addition. All hashtags (e.g. #happy) were split into two tokens: one token for the hashtags sign (i.e. #) and the other token for the word (i.e. happy). Hence, the same embedding was used regardless if the word is a part of the sentence or whether it is inside a hashtag. Any potential difference in meaning is encapsulated in the embedding for # character that comes just before the word itself.

Since we have a lot of data and since our task differs from the unsupervised task used to train GloVe word embedding, we also tested whether the fine-tuning of word embedding during the training of our networks would improve the performance. This, along with the four different dimensionalities of embeddings provided by GloVe, were considered as the parameters we tuned:

- GloVe embedding dimensions: 25, 50, 100, 200;
- fine-tune GloVe embedding: no, yes;

For our character-based approach, we trained character embedding from scratch, starting from a random initialisation, since we were not aware of any publicly available embedding. Our dictionary contained a set of all characters that appeared at least 25 times across all our data sets. That yielded a set of 410 different characters, including emoticons and symbols, for which we trained the embedding. We searched among these dimensionalities:

- character embedding dimensions: 10, 15, 25, 35;

In our character-based approach, we left the tweet's content almost untouched. We only removed the hashtags used to set the target category and URLs due to the anomaly of our crawler. If the URLs on Twitter are mostly shortened (e.g. *pic.twitter.com/<hash>*), our crawler has already expanded them which made some tweets drastically longer than 140 characters. Hence, we decided to remove all URLs from tweet contents' for our character-based experiments.

4.2.2 Recurrent Neural Networks

The architecture of our recurrent neural networks was the following. The sequence of either words or characters was first passed through the dropout layer to combat overfitting. Next, one or more recurrent layers followed with a dropout layer in between any two pairs of consecutive recurrent layers. We experimented with LSTM and GRU types of recurrent cells, different dimensionalities of hidden states, bidirectional layers, and different dropout rates. After the final hidden state of the recurrent layer and the final dropout layer, a dense layer was used with either softmax (for multiclass setting) or sigmoid activation (for multilabel setting). The architecture of our recurrent networks is depicted in Figure 4.1 left.

We searched for an optimal parameter set among the following configurations:

- layer kind: LSTM, GRU;
- hidden layers: 1, 2, 3;
- neurons per hidden layer: 200, 500, 1000, 2000;
- bidirectional layers: no, yes;
- dropout for word or character embedding: 0.0, 0.2;
- dropout between multiple layers: 0.0, 0.2;
- dropout for softmax (sigmoid): 0.0, 0.5.

The dropout rates for softmax come from Srivastava et al. [76] and for embedding from Zaremba et al. [77]. We used RMSProp optimiser [25] for RNNs, a batch size of 128, and an early stopping with the patience of five. All RNN experiments were conducted in Python using Keras [78] library and Theano [79] backend.

4.2.3 Convolutional Neural Networks

For convolutional neural network experiments, we followed Kim's architecture [21] that is presented in Section 2.3.3. That is, after embedding the input, one convolution and max-pooling over time layers followed before the final dense layer. The architecture of our convolutional networks is presented in Figure 4.1 right.

We searched for an optimal parameter set among the following configurations:

- feature maps: 100 – 6000 (with step 100)
- feature maps activation: relu, tanh
- kernel size: 1 – 20
- dropout for word or character embedding: 0.0, 0.2;
- dropout for softmax: 0.0, 0.5.

We started the parameter search in the ranges suggested by Zhang & Byron [24], which turned out to be appropriate for word-based experiments. For character ones, however, the kernel sizes had to be enlarged to capture the large enough context of the tweet to support recognising words. Also, due to the more complex task character-based networks have to tackle, more feature maps were required.

We used Adam optimiser [28] for CNNs, a batch size of 128, and an early stopping with the patience of five. Similarly to RNNs, all experiments were conducted using Keras [78] library and Theano [79] backend.

4.3 *Transfer Learning*

After the best neural network models were selected along with their parameters, we investigated the generality and transferability of learned representations. More specifically, we are interested whether the final hidden state representations — which can be considered as a projection of the whole tweet's content into a low-dimensional space — are informative enough to support the recognition of other emotion categorisations, not just the one they were trained for. To explore this, we designed a set of experiments that copy some neural network parameters from one task to the other. We took all parameters that influence the values at the final hidden state trained on a first data set, copied, and froze them in a new neural network that was then trained for another task. Doing so, we forced the neural network corresponding to the second task to use the projection into the final hidden state learned during the first task. During the training of the second task, only the weights of the final dense layer were modified. Hence, the neural network for the second task can only work on top of features learned by the first one. Notice that since we were copying parameters from one network to another, both networks needed to have the same architecture; i.e. the number of neurons, number of layers, type of

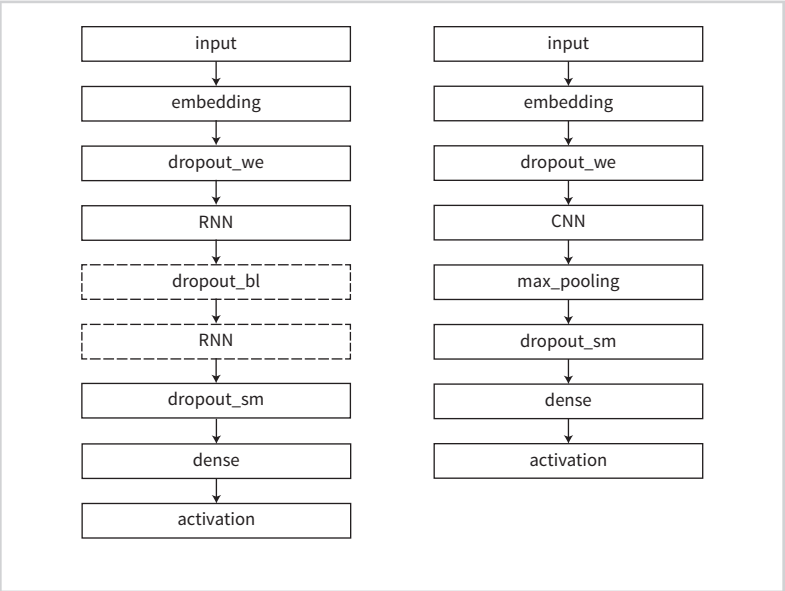


Figure 4.1
The architecture of our RNN (left) and CNN (right) models. Dashed boxes are only present in multi-layer architectures. The RNN figure corresponds to a two-layer architecture, while the three-layer architecture includes another pair of *dropout_bl* and *RNN* layers.

layers, number of feature maps, kernel size, etc. The graphical representation of these experiments is depicted in Figure 4.2.

The performance of the neural network on the second task was then compared with the performance for that task when the neural network was trained from scratch while allowing the updates of all parameters. There are two possible outcomes of these experiments. If the performance is comparable, this indicates that features discovered during the training of the first task are general enough to support the prediction of the second task as well. Hence, this would demonstrate that the projection of the tweet into lower dimensional representation is suitable for recognising other emotion classifications. We could consider the projection as a universal lower-dimensional emotional representation of the tweet. Contrary, if the performance drops drastically, this would demonstrate that features trained during the first task are not directly useful for the second task.

These transfer experiments results shed light on the generality of features when neural networks were trained for each task separately, which is an essential prerequisite before developing a unison model. If the features discovered during the training on one task are

general enough to support others, then unison model is unnecessary. We can just take the projection into the final hidden state and train simple classifiers on top, to make predictions for other tasks as well. Contrary, if the results of transfer experiments were poor, that would make a unison model a candidate for improving the generality of final hidden state representations.

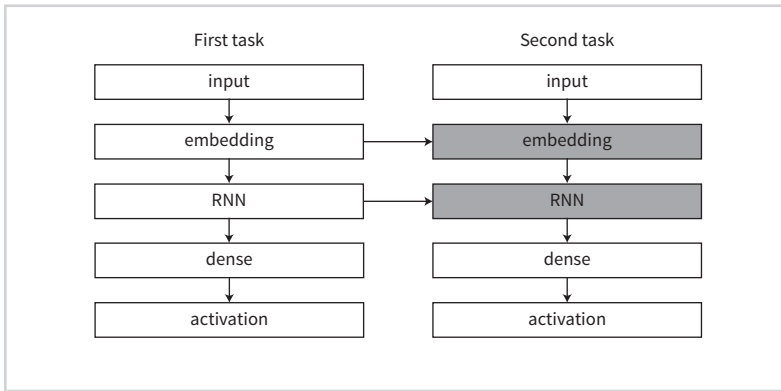


Figure 4.2
The schema of transfer experiments for RNNs. We first trained the architecture for the first task and then copied the weights to the second task, as indicated by horizontal arrows. The greyed out layers in the second task indicate the layers which were fixed after initialisation. Hence, when training the second task, the model was forced to use the weights as they were learned during the first task.

4.4 Unison Learning

Our final set of experiments focused on the development of a unison model — one single model for recognising all three emotion classifications. The architecture of our unison model was the following. We started with a shared word or character embeddings that were followed by shared RNN or CNN layers. In the case of multilayer architectures for RNNs, all recurrent layers were shared among tasks. After the final hidden state, multiple task-specific shallow layers followed. For each emotion classification task, we had one dense (fully connected) layer connecting the shared representation to the output neurons. On them, we applied either softmax or sigmoid activation to obtain the model output for the specific task. The architecture of our unison models is presented in Figure 4.3. Note that we built on the ideas presented by Collobert & Weston [30]. However, their tasks were not as closely related as ours and hence they only shared word

embedding, while all other neural networks layers were task specific. As all our tasks tackle emotion recognition, we restricted our unison models even more by forcing them to share other neural network layers as well. The only task-specific layers were the final outputs, which cannot be shared between different tasks.

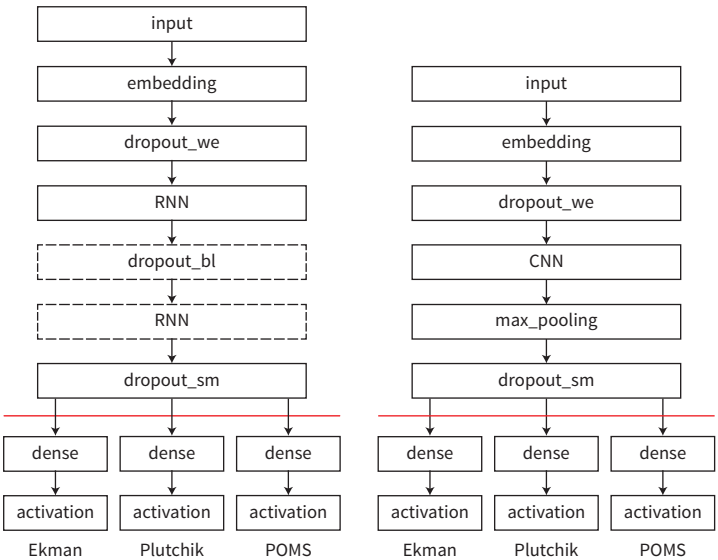


Figure 4.3
The architecture of RNN (left) and CNN (right) unison models. Everything above the red line was shared for three emotion classifications; i.e. models used the same word or character embedding as well as the whole RNN or CNN layer. On top of the common representation, one final dense layer followed with either softmax (or sigmoid) activation for each emotion classification.

This parameter sharing architecture forced the unison model to discover final hidden state representations that can support recognising multiple emotion classifications. Hence, if the model in such a setting can perform well, then the final hidden state representation can be considered a general emotion representation of the input content. As such, it could be a great starting point for investigating the interdependence or correlation between different emotional classifications. Also, similarly to pre-trained word embedding for our models, this general representation could be a great starting point when training models for other emotion classifications not considered in our study.

As we possess three separate data sets with emotion labelled tweets, we could not adopt the gradient summation during the training of our models. Hence, we trained

our models with the approach presented by Collobert & Weston [30], which we refer to as *alternating batches* strategy. Initial experiments with the alternating batches strategy showed that it is not suitable for our setting with drastically different data set sizes. Recall that our largest data set is more than ten times larger than our smallest one. Hence, we proposed two novel training heuristics that were designed to better handle training of unison models with different data set sizes or complexities: *weighted sampling batches* and *weighted sampling batches by data set size*. These three training heuristics, the known alternating batches and the two newly proposed ones, are presented in the following three subsections.

4.4.1 Alternating Batches

The *alternating batches strategy (AB)*, presented by Collobert & Weston [30], trains the unison model by iterating over the training sets and each time picking a random example from the currently selected training set. The neural network weights are updated according to gradients with respect to this example. Note that only shared parameters and the parameters corresponding to the current task are updated. The parameters corresponding to other tasks are left unchanged. After the parameters update, the training heuristic proceeds by picking the next task.

Instead of working with only one example at a time, we used batches of multiple examples at each step. Also, we introduced early stopping into the training iteration to finish the training before the network started to overfit. As we were working with multiple data sets, the early stopping monitored the average validation accuracy across all three data sets. The alternating batches strategy is presented in Algorithm 2.

Note that all data sets on input have already been split into training and validation part. The function *next_train_batch(x)* returns the next training batch for data set *x* and the function *train_on_batch(b, m)* performs one gradient-based parameters update of the model *m* with respect to the training examples from batch *b*.

At the end of each epoch, the early stopping validation criterion is checked and the training is aborted when the criterion is met.

4.4.2 Weighted Sampling Batches

We proposed a novel *weighted sampling batches (WSB)* training heuristic to improve the alternating batches strategy for cases where data sets differ by size or complexity. Instead of devoting the same amount of attention during the training process to all data sets,

Algorithm 2 Alternating Batches strategy by Collobert & Weston [30].

Input: DS = $\{d_1, d_2, \ldots, d_n\}$ ▷ data sets
MODEL ▷ initialised NN model
EPOCHS ▷ max number of epochs
UPDATES ▷ number of updates in epoch
Output: MODEL ▷ trained NN model

```
for epoch = 1 → EPOCHS do
    for update = 1 → UPDATES/|DS| do
        for ds ∈ DS do
            b ← next_train_batch(ds)
            train_on_batch(b, MODEL)
    for ds ∈ DS do
        /* evaluate model on train and validation set */
    if early stopping criterion met then
        break
```

WSB monitors the progress of each data set separately. It then adapts the training to take more examples from data sets where the progress is slower. Doing so, it allows the network to focus more on the examples coming from harder data sets.

We estimate the progress with a technique that is commonly adopted by researches when training neural networks — i.e. observing the accuracy of the model through the training iterations. At the end of each epoch, we first calculate the train and validation accuracy of the unison model for each of the data sets. When the model starts to overfit a certain data set, the difference between the train and validation accuracy starts to grow as the model becomes better and better on the train part while on validation part its performance stagnates or drops. Hence, we believe that the difference between the train and validation accuracy, which is small at first and becomes larger the closer we are to the point of overfitting, is indicative of the training progress for a given data set. We treat the differences between the train and validation accuracies as our progress estimates, and use them to steer the sampling process of the next training batch. Instead of sampling the training batches uniformly from each data set (as in the alternating batches strategy) we sample the next training batch with weights based on our progress estimates.

By sampling more batches from the data sets where the progress is slower, we boost the fitting to those data sets. The sampling probabilities are inversely proportional to the progress estimates[2]. Hence, the higher the difference between the train and validation accuracy for a given data set, the fewer batches will be sampled from it. As progress estimates are only available after the first epoch, we initialise the sampling probabilities to be equal for all data sets. Doing so, the algorithm devotes an equal amount of attention to all data sets in the first epoch. Then after each epoch, the sampling probabilities are recalculated according to the current progress estimates. Notice that if the heuristic estimates the progress on all data sets to be equal, the sampling probabilities would also be equal, and the WSB strategy would degenerate into the AB strategy. The complete WSB strategy is presented in Algorithm 3.

Note that the function *random_choice(DS, weights)* samples a random data set out of *DS* weighted by sampling probabilities given as *weights*. Functions *train_acc(ds)* and *val_acc(ds)* calculate the accuracy on a train and validation part of the data set *ds* respectively.

Finally, notice that the WSB heuristic requires a labelled validation set. In the parameter selection phase, we run it on the train set and used the validation set to establish sampling probabilities. Once all other model parameters were set (e.g. number and dimensionality of the layers) we needed to select the sampling probabilities to be used when the classifier was trained for the final evaluation on the test set. Then, we are not allowed to observe the accuracy after each epoch to set sampling probabilities as we did in the parameter selection phase. If we would, the final evaluation would not be fair. Hence we needed to provide the sampling probabilities throughout all epochs on input[3]. We decided to calculate average sampling probability through all epochs, discarding the initial weight which was set to $1/n$ at the beginning of the algorithm, and used that average as the sampling probability through all epochs when the model was trained in the test mode. We emphasise that in the test mode, when the model was trained on the combination of train and validation sets, the algorithm did not evaluate the progress after each epoch as presented in Algorithm 3 and the class labels corresponding to testing

[2] Notice that by making the sampling probabilities inversely proportional to the progress estimates, it could happen that some data sets would not get chosen at all. If this becomes problematic, we could change the algorithm slightly to enforce lower bounds on sampling probabilities for each data set and treat this as another parameter of the algorithm.

[3] Similarly as the number of epochs determined by early stopping is provided for final evaluation.

Algorithm 3 Proposed Weighted Sampling Batches strategy.

Input: DS = $\{d_1, d_2, \ldots, d_n\}$ ▷ data sets
MODEL ▷ initialised NN model
EPOCHS ▷ max number of epochs
UPDATES ▷ number of updates in epoch
Output: MODEL ▷ trained NN model

```
weights ← [1/n, 1/n, ..., 1/n]
for epoch = 1 → EPOCHS do
    for update = 1 → UPDATES do
        ds ← random_choice(DS, weights)
        b ← next_train_batch(ds)
        train_on_batch(b, MODEL)
    for ds ∈ DS do
        /* evaluate model on train and validation set */
        progress ← train_acc(ds) − val_acc(ds)
        weights[ds] ← 1/progress
    weights ← weights/sum(weights)
    if early stopping criterion met then
        break
```

examples were not revealed to the algorithm. We consider the sampling probabilities as another parameter of the model that needs to be set before evaluating it in the test mode.

The reason for using the fixed sampling probabilities through all iterations when training the model in the test run is that our sampling probabilities nicely converged (see Figure 5.2 in Section 5.4). If the sampling probabilities plot in the parameter selection phase were not so flat, we could skip the averaging and directly use the corresponding sampling probabilities as set in each epoch by the algorithm in the parameter selection phase.

4.4.3 Weighted Sampling Batches by Data Set Sizes

To test the utility of the progress estimation as defined in WSB heuristic, we defined another simpler heuristic to compare it with. Instead of using progress estimates to steer

the sampling of batches, we sample with fixed sampling probabilities throughout all epochs of the training algorithm. The sampling probabilities are set according to the data set sizes, by which we achieve that each training example will be used approximately the same number of times for updating network parameters. Consequently, the estimation of the model performance on train and validation set after each epoch is not needed any more. The *weighted sampling batches by data set sizes (WSBDS)* heuristic is presented in Algorithm 4.

Algorithm 4 Proposed Weighted Sampling Batches by Data Set Sizes strategy.

Input: DS = $\{d_1, d_2, \ldots, d_n\}$ ▷ data sets
MODEL ▷ initialised NN model
EPOCHS ▷ max number of epochs
UPDATES ▷ number of updates in epoch
Output: MODEL ▷ trained NN model

$weights \leftarrow [|d_1|, |d_2|, \ldots, |d_n|]/sum([|d_1|, |d_2|, \ldots, |d_n|])$
for $epoch = 1 \rightarrow$ EPOCHS **do**
    for $update = 1 \rightarrow$ UPDATES **do**
        $ds \leftarrow random\_choice(\text{DS}, weights)$
        $b \leftarrow next\_train\_batch(ds)$
        $train\_on\_batch(b, \text{MODEL})$
    if early stopping criterion met **then**
        break

5

Results and Discussion

Our experimental results are split into seven sections. We first present the results of experiments with traditional text classification approaches on single data sets in Section 5.1. These set the baseline for comparison with neural networks, whose performance is reported in Section 5.2. Next, we investigate the transfer capabilities of our neural networks' final hidden state representations in Section 5.3, which gives intuition on the generality of the learned representations. The results of our unison models, along with the comparison of alternating batches heuristics with the newly proposed weighted sampling batches and weighted sampling batches by data set sizes heuristics, are presented in Section 5.4. As our thesis focuses on emotion recognition, we evaluated the proposed training heuristics only in this context and did not focus on how they perform on other, unrelated tasks to which they may be applicable. In Section 5.5 we demonstrate the utility of applying the trained embedding to other data sets, while in Section 5.6 we compare different emotion classifications used in this study. We conclude the chapter by discussing ideas for future work along with the study's limitations in Section 5.7.

Before discussing the results, we emphasise that all model parameters, like those corresponding to the architecture of neural networks (e.g. number of layers, dimensionality of the layers) as well as those corresponding to their training (e.g. number of epochs to train, sampling probabilities for training unison models with our heuristics) were selected according to the model performance on validation set when the models were trained on the train set. With all the parameters set, we trained the models on the combination of train and validation sets. These models were then finally evaluated only once on the test set. Notice that the test set true labels were never revealed to any of the algorithms during the parameter selection phase and that duplicate detection and re-tweet removal, as described in Chapter 3, assured that none of the tweets in the test set was present in the train or validation set.

We report the performance on the test set using macro- and micro-averages of F1-score, which are defined as follows. For a tweet and an emotion category, there are four distinct outcomes, as illustrated in Table 5.1, that differ based on the fact whether the emotion is expressed in the tweet or not, and whether the classifier recognised it or not. Given the numbers of tweets belonging to each of the above cases, we calculate the *precision* in Equation 5.1, *recall* in Equation 5.2, and *F1-score* in Equation 5.3.

$$\mathrm{P}\mathit{recision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{5.1}$$

Table 5.1
The confusion matrix for a given emotion and the corresponding classifier's output. The emotion can either be expressed in the tweet or not, and the classifier can either recognise it or not.

		Emotion expressed in text?	
		Yes	No
Emotion recognised?	Yes	True Positive (TP)	False Positive (FP)
	No	False Negative (FN)	True Negative (TN)

$$\text{R}ecall = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5.2}$$

$$\text{F1}\text{–}score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5.3}$$

The distinction between a *micro-averaged* and *macro-averaged* F1-score is where in the above equations do we aggregate across multiple categories. For a *macro-averaged* F1-score we first calculate the F1-scores for each emotion category separately and then average those to obtain an overall score. Doing so, the macro-averaged F1-score treats all classes equally.

Contrary, a *micro-averaged* F1-score first sums TP, FP, FN, and TN cases across all categories and uses those to calculate the overall F1-score. Hence, each training example makes an equal contribution to the overall score.

5.1 Traditional Text Classification

Results of traditional text classification experiments on bag-of-words models are presented in Table 5.2 for the multiclass mode and in Table 5.3 for the multilabel mode.

We observed that normalisation of unigrams did not improve the performance for SVM, NB, and LogReg, but it brought noticeable improvements for RF on unigrams. When training models on the combination of unigrams and bigrams, the token normalisation shows as more beneficial. The performance on bigrams is almost always better when using normalisation for all classifiers in both multiclass and multilabel mode. Comparing the performance of classifiers when trained on only unigrams with the performance when trained on a combination of unigrams and bigrams, the addition of bi-

Table 5.2

F1-scores (macro/micro) of different classifiers on different bag-of-words models in the multiclass mode as measured on the test set. For each of the averaging techniques, the best performance for each data set is underlined.

		Multiclass			
		Unigrams		Bigrams	
		Vanilla	Normalised	Vanilla	Normalised
Ekman	SVM	58.5 / 69.4	58.1 / 69.5	59.5 / 70.6	60.2 / 71.2
	NB	56.0 / 67.3	56.4 / 67.1	55.6 / 68.1	56.2 / 68.4
	LogReg	58.7 / 69.5	58.7 / 69.7	59.2 / 70.8	60.1 / 71.5
	RF	46.8 / 64.6	47.5 / 65.6	47.5 / 65.1	47.7 / 65.8
Plutchik	SVM	54.4 / 66.8	54.1 / 66.7	56.3 / 68.2	57.1 / 69.0
	NB	51.8 / 64.5	52.2 / 64.3	52.5 / 65.6	53.0 / 66.0
	LogReg	54.7 / 67.0	54.8 / 66.9	56.0 / 68.5	57.0 / 69.3
	RF	41.7 / 61.8	42.8 / 62.7	43.0 / 62.4	43.5 / 63.3
POMS	SVM	63.8 / 66.8	63.5 / 66.8	67.5 / 71.0	68.2 / 71.5
	NB	61.0 / 64.4	60.9 / 64.4	63.9 / 66.9	64.2 / 67.3
	LogReg	64.2 / 67.0	64.0 / 67.0	68.1 / 71.2	68.8 / 71.7
	RF	57.6 / 64.0	59.1 / 65.2	57.2 / 63.9	59.1 / 65.3

grams brings more contextual information, which is beneficial in all cases; except for RF on Ekman and Plutchik in multilabel mode. This is understandable, as the latter is the superset of features of the former. Interestingly, the advantage of bigrams over just unigrams gets larger on larger data sets. For LogReg on Ekman, the absolute improvement is up to 2 % while for POMS it is up to 5 % in absolute.

When comparing different classifiers, the one that consistently shows drastically worse performance than all others in all settings is RF while it also requires substantially more time to train. We believe the primary reason for this is the enormous feature space dimensionality of the bag-of-words models along with its sparseness. Notice that despite the reduced parameter search for POMS, the results seem comparable to the other two data sets. We observed only slight improvements on Ekman and Plutchik when going from 200 trees to up to 5000 and tuning the number of features. Hence, we have not

Table 5.3

F1-scores (macro/micro) of different classifiers on different bag-of-words models in the multilabel mode as measured on the test set. For each of the averaging techniques, the best performance for each data set is underlined.

		Multilabel			
		Unigrams		Bigrams	
		Vanilla	Normalised	Vanilla	Normalised
Ekman	SVM	55.5 / 66.8	54.8 / 66.7	56.4 / 68.2	56.4 / 69.2
	NB	55.1 / 65.8	55.4 / 65.7	55.2 / 66.7	55.8 / 66.8
	LogReg	55.5 / 67.1	55.7 / 67.3	56.3 / 68.5	57.4 / 69.3
	RF	39.2 / 56.3	39.8 / 57.3	37.0 / 56.1	37.2 / 57.0
Plutchik	SVM	50.3 / 63.2	49.9 / 62.9	52.5 / 65.3	53.4 / 66.3
	NB	50.4 / 62.6	51.0 / 62.5	50.8 / 63.1	51.4 / 63.3
	LogReg	51.3 / 63.9	51.0 / 63.8	52.7 / 65.5	53.3 / 66.6
	RF	32.4 / 50.7	33.6 / 52.6	29.7 / 49.2	30.2 / 50.7
POMS	SVM	59.0 / 61.4	58.4 / 61.1	65.1 / 67.9	65.9 / 68.5
	NB	58.3 / 62.7	58.3 / 62.6	60.3 / 65.5	60.5 / 65.8
	LogReg	60.3 / 62.6	60.0 / 62.4	65.3 / 68.3	66.3 / 69.0
	RF	44.4 / 51.5	46.0 / 53.0	47.3 / 56.5	49.1 / 57.7

attempted a full parameter search and, in particular, a larger number of trees on POMS because it would take too long. Although the performance of the remaining three classifiers is almost comparable, LogReg seems slightly better than SVM, which in turn seems slightly better than NB. The best overall results were always achieved with LogReg when trained on normalised bigrams. This indicates that the addition of bigrams and our token normalisation strategy were beneficial to the classifier. Classifiers in multilabel mode achieved a slightly worse performance than classifiers in multiclass mode, which is understandable as the multilabel approach tackles a harder problem.

Comparing macro- and micro-averaged scores reveals the importance of adequately large training data set. For Ekman and Plutchik, the differences are in the range of 10-15 % in absolute with the micro-averaged score always being higher than the macro-averaged one. For POMS these difference are usually below 7 % also in favour of micro-averaging.

Recall that in the macro-average each class contributes equally to the overall score, regardless of the number of training examples inside it. Contrary, in the micro-average all training examples contribute equally to the final score disregarding of the class they belong to. Hence, we believe that for Ekman and Plutchik our classifiers handle the large categories well but perform poorly for categories with few examples. For POMS, as the difference between macro- and micro-averages is smaller, we believe the performance of classifiers is better even for classes with few training examples. As class distributions are imbalanced for all three data sets, we are attributing the better handling of examples from minority classes to the drastically larger data set size for POMS.

Results of experiments with latent semantic indexing models are reported in Table 5.4 for the multiclass mode and in Table 5.5 for the multilabel mode. Recall that experiments were only performed for Ekman and Plutchik as calculating SVD for POMS was infeasible.

Table 5.4

F1-scores (macro/micro) of different classifiers on different latent semantic indexing models in the multiclass mode as measured on the test set. For each of the averaging techniques, the best performance for each data set is underlined.

		Multiclass			
		Unigrams		Bigrams	
		Vanilla	Normalised	Vanilla	Normalised
Ekman	SVM	40.5 / 57.9	39.2 / 57.0	53.4 / 66.9	53.2 / 67.0
	NB	31.8 / 41.0	32.7 / 42.2	22.5 / 37.0	24.7 / 38.1
	LogReg	42.2 / 58.5	40.8 / 57.5	54.5 / 67.0	54.5 / 67.2
	RF	36.9 / 55.6	38.1 / 56.7	32.5 / 53.3	36.1 / 55.3
Plutchik	SVM	34.0 / 53.8	32.7 / 52.7	49.5 / 63.9	49.1 / 64.0
	NB	25.9 / 35.4	26.6 / 36.3	16.7 / 30.4	18.5 / 31.3
	LogReg	36.4 / 54.4	34.5 / 53.3	50.8 / 64.2	50.9 / 64.3
	RF	31.8 / 52.7	32.2 / 53.1	28.7 / 50.7	31.4 / 52.7

The trends we observed are similar to those on bag-of-words models. Normalisation on bigrams seems to help most of the time, while on unigrams it is only beneficial for RF and NB. The addition of bigrams only improved the performance when using the SVM

Table 5.5

F1-scores (macro/micro) of different classifiers on different latent semantic indexing models in the multilabel mode as measured on the test set. For each of the averaging techniques, the best performance for each data set is underlined.

		Multilabel			
		Unigrams		Bigrams	
		Vanilla	Normalised	Vanilla	Normalised
Ekman	SVM	33.3 / 49.5	31.5 / 47.8	48.2 / 63.2	47.8 / 63.2
	NB	33.4 / 39.6	34.4 / 41.4	27.9 / 33.2	29.0 / 34.5
	LogReg	35.7 / 51.6	33.9 / 49.9	51.2 / 64.3	51.1 / 64.5
	RF	26.5 / 37.8	27.9 / 40.1	18.8 / 29.2	23.8 / 34.5
Plutchik	SVM	25.1 / 42.8	23.1 / 40.6	42.5 / 58.8	41.8 / 58.7
	NB	26.8 / 30.1	27.3 / 31.4	23.7 / 26.7	24.6 / 27.8
	LogReg	29.0 / 45.7	26.6 / 43.5	46.5 / 60.5	46.1 / 60.6
	RF	21.1 / 34.5	21.4 / 35.9	15.8 / 27.0	18.7 / 31.7

or LogReg classifier, while it hurts the performance for NB and RF. The performance of RF is again inferior to SVM, NB, and LogReg which again show comparable performance. The best overall results were once again achieved using LogReg on normalised bigrams, confirming the usefulness of adding bigrams and normalising tokens.

The differences between macro- and micro-averages — although the scores are still in favour of micro-averaging — grew larger, even up to 20 % in absolute, indicating that LSI performance for small classes is worse than that of bag-of-words models. As the performance of LSI models is considerably worse than those of bag-of-words models — across classifiers, n-grams, and token normalisation techniques — we use the LogReg trained on the bag-of-words model with normalised bigrams as our baseline for comparison with neural networks.

Finally, comparing all three emotion classifications, the accuracy of predicting POMS's and Ekman's categories is comparable. With our approach of labelling tweets, the higher complexity of POMS is counterbalanced by the much larger number of training tweets. The accuracy of predicting Plutchik's categories is a bit lower, which is expected as it contains two more categories than Ekman or POMS.

5.2 Neural Networks

Before presenting the results, we discuss some general observations. When investigating the GloVe embedding and comparing it with the fine-tuned version, we observed that GloVe embedding is too-focused on word co-occurrences to be directly suitable for emotion recognition task. For example in GloVe, the most similar word, in terms of cosine similarity, to word *sadness* is *happiness*; and to word *angry* is *birds*. After our fine-tuning, the closest word to *sadness* is *depression*; and to *angry* it is *annoyed*. Considering these examples, we believe that although the GloVe embedding initially was not able to distinguish between detailed differences in word meaning crucial for emotion recognition, the process of fine-tuning made it much more suitable for the task. This suspicion is also confirmed by the fact that the performance of our models was always better if we allowed the model to fine-tune the word embedding. Hence, we conclude that the fine-tuning of word embedding is beneficial. Regarding the embedding size, larger is better. In seven out of eight cases the 200-dimensional embedding yielded the best results, while in the eighth case the difference between 100- and 200-dimensional embedding was negligible.

Character embedding dimensionality can indubitably be much smaller, as it only represents a set of 410 different characters and not the meaning of 1.2M words. Although we selected 25-dimensional representation, using only 10-dimensional one did not seem to harm the performance much; in some cases, the decrease in performance was less than 1% in absolute.

Regarding the network architecture of RNNs, we observed that LSTM and GRU layers performed comparable, while initial experiments with the simple recurrent layers showed inferior performance and hence we did not include it as an option in the parameter search phase. Adding more recurrent layers only occasionally yielded minor improvements; similarly using bidirectional layers did not bring any improvements when working with words on input. However, when working with character input, the bidirectional layers almost always helped to improve the performance.

For CNNs on words, the best kernel sizes are between 2–3 and the number of feature maps in the range from 1500–2000, which is a bit larger than suggested for CNN text classification architectures [24]. When working with character input, such kernel sizes and number of feature maps were insufficient. Intuitively, as the words on input have been replaced with characters, one needs to enlarge the window size to capture large enough context to support recognition of words. As experiments showed, the larger

Table 5.6

F1-scores (macro/micro) for RNNs and CNNs in the multiclass mode as measured on the test set. For easier comparison, we also show the best BoW performance. The best micro or macro score for each data set is underlined.

		Multiclass		
	Input	RNN	CNN	*Best BoW*
Ekman	Word	60.7 / 71.4	59.9 / 69.2	*60.1 / 71.5*
	Char	60.4 / 71.2	60.1 / 71.8	
Plutchik	Word	52.8 / 65.5	55.1 / 66.8	*57.0 / 69.3*
	Char	57.5 / 70.0	57.0 / 69.2	
POMS	Word	67.9 / 70.4	67.1 / 69.9	*68.8 / 71.7*
	Char	70.5 / 73.2	68.1 / 70.9	

kernel size by itself is not sufficient, but the number of features maps needs to be enlarged as well. In our character-based CNN experiments we used kernel sizes of 9–11 and the number of feature maps between 5000–6000.

Results of experiments with neural networks along with the best bag-of-words models to ease the comparison are summarized in Tables 5.6 for multiclass and Table 5.7 for the multilabel mode.

The success of character approach seems dependent on the training data set size. For POMS, which is the largest of our data sets, networks always achieved better performance on characters than on words. For Plutchik, the second largest, this happened in most cases and for Ekman in some. Predicting directly from characters is more challenging as the network needs to discover the concept of words by itself. Hence, we believe that when sufficient training data is available, it is best to opt for character-based NN approach. Contrary, when the training data is scarce, we suggest to work on words and rely on the pre-trained embedding with some fine-tuning. However, this comparison is not entirely fair. For word approaches, we used GloVe as our initialisation, which already defined meaningful representations of words which we just fine-tuned further to incorporate sentiment. For characters, we started learning their representations from random initialisations, which contain no background knowledge. We suspect that if pre-trained

Table 5.7

F1-scores (macro/micro) for RNNs and CNNs in the multilabel mode as measured on the test set. For easier comparison, we also show the best BoW performance. The best micro or macro score for each data set is underlined.

		Multilabel		
	Input	RNN	CNN	*Best BoW*
Ekman	Word	57.3 / 70.0	60.2 / 69.8	*57.4 / 69.3*
	Char	56.4 / 70.3	54.9 / 68.9	
Plutchik	Word	55.5 / 67.7	56.9 / 67.0	*53.3 / 66.6*
	Char	52.6 / 68.6	53.9 / 66.8	
POMS	Word	67.0 / 70.4	65.7 / 68.8	*66.3 / 69.0*
	Char	67.5 / 70.7	66.7 / 69.0	

character embedding would exist when we designed our experiments[1], using it instead of random initialisation might work in favour of the character approach.

When comparing RNNs with CNNs we observed that in the vast majority of cases RNNs outperformed CNNs, though not by a considerable margin. Further, when comparing the best NN approach with the best BoW approach, we saw that NN always gave slightly better results that BoW. Similarly, as for bag-of-words, Ekman and POMS showed comparable performance while the performance of Plutchik was a bit worse due to more categories. Discrepancies between macro- and micro-averaged scores were the same as for BoW — larger for Ekman and Plutchik and smaller for POMS.

To explicitly answer our first guiding question of the study: neural networks achieve slightly better performance than traditional text classification approaches; although, the differences are not nearly as high as we hoped for. However, there are also other benefits of using neural networks, as in the character setting they do not require any human intervention and language-specific prerequisites. Recall that the best BoW scores were achieved with the bigrams on normalised words and that performance was worse without token normalisation and bigrams. Contrary, character-based NNs do not require any tokenisation, preprocessing, token normalisation, stemming or filtering as

[1] When finalising the thesis, after all experiments were already completed, a character embedding BERT [80] became available.

networks work directly on the stream of unprocessed text characters. Hence, since no language-specific prerequisites and no feature engineering is required, character-based neural networks are more appropriate even for low-resource languages.

These improvements, however, come at the expense of higher computational costs. While training of LogReg on POMS took up to 13 hours, word-based RNNs took up to 5 days and character-based ones up to 8 days on a single GPU. CNNs, due to much simpler architecture, took from a few hours up to one day.

5.3 Transfer Learning

Results of the transfer experiments are reported in Table 5.8 for multiclass and Table 5.9 for the multilabel setting. Notice that transfer experiments require a common architecture across all data sets to support copying of parameters from one network to the other. Hence, when comparing the transfer results performance metrics, they cannot be compared with results from Section 5.2. Instead, they should be compared with the results on this common architecture, which we report along the diagonals of the Tables 5.8 and 5.9.

We observed that the only setting in which transfer experiments gave comparable performance to training a single model is when the initial model was trained on Plutchik and applied to Ekman. In some cases, this yielded even better performance than training on Ekman itself. Recall that Ekman's categories are a subset of Plutchik's and so are all train, validation, and test sets (see Section 3.2). Hence, when training the model on Plutchik and then transferring those weights to Ekman, the only thing that differed, in comparison to training on Ekman directly, is the presence of two additional categories *trust* and *anticipation*. It is known for multi-task learning [29] that additional categories during training can introduce more training signal and lead to better generalisation.

Transfer result in the other direction, i.e. training the initial model on Ekman and transferring the weights to Plutchik, was noticeably worse. We suspect that the absence of *trust* and *anticipation* categories during the time of initial model training prevented the discovery of required features for their recognition, which consequently led to inferior performance on these two categories. Observing the confusion matrices reveals the following. First, the transfer model rarely predicts the *anticipation* category and frequently confuses it with the most common Ekman's category *surprise*. Second, the transfer model confuses all categories with the largest Plutchik's category (*trust*) much more frequently than the single model.

Table 5.8

Transfer experiments F1-scores (macro/micro) for word and character approach in the multiclass settings on the test set. The table consists of four 3x3 matrices each corresponding to a set of transfer experiments in one setting. The values on the diagonals correspond to the results obtained when training the data set on a common architecture. All the off-diagonal values, showing the transfer results, should be compared with the corresponding diagonal one. The best micro or macro score for each data set is underlined.

				Softmax Tuning and Testing Multiclass		
				Ekman	Plutchik	POMS
Initial Weight Learning	RNN	Word	Ekman	57.9 / 68.9	44.9 / 59.6	37.1 / 49.1
			Plutchik	57.7 / 69.1	54.1 / 66.0	37.3 / 49.3
			POMS	51.5 / 63.0	48.0 / 59.7	68.5 / 71.3
		Char	Ekman	60.3 / 71.8	46.5 / 60.5	36.5 / 49.2
			Plutchik	61.1 / 72.4	58.5 / 69.9	39.5 / 50.9
			POMS	37.6 / 51.1	31.9 / 45.5	69.9 / 72.8
	CNN	Word	Ekman	59.9 / 70.1	51.7 / 62.7	52.9 / 57.8
			Plutchik	59.3 / 69.4	56.0 / 66.9	53.6 / 57.9
			POMS	45.6 / 56.3	39.1 / 51.2	66.7 / 69.8
		Char	Ekman	57.4 / 69.6	50.0 / 64.0	48.7 / 58.0
			Plutchik	60.0 / 71.5	54.3 / 68.0	50.5 / 59.2
			POMS	49.3 / 62.7	44.8 / 58.2	65.8 / 69.5

Due to the similarity between Ekman and Plutchik, the transfer between them cannot be considered a real transfer experiment as some of the categories were present during the time of initial model training. As this is not true when transferring from either Ekman or Plutchik to POMS, or the other way around, investigating these results gives more insight into the real transfer capabilities of our models. When training the initial RNN model on Ekman and transferring it to POMS, we observed that the best micro-averaged F1-score was around 58 % while the worst one was around 20 %. Comparing these to the 70 % score when training POMS on common architecture revealed that restricting the POMS's model to work on Ekman's hidden state representation resulted in a significant performance drop. Similar trends hold for other RNN transfers as well: from Plutchik

Table 5.9

Transfer experiments F1-scores (macro/micro) for word and character approach in the multilabel settings on the test set. The table consists of four 3x3 matrices each corresponding to a set of transfer experiments in one setting. The values on the diagonals correspond to the results obtained when training the data set on a common architecture. All the off-diagonal values, showing the transfer results, should be compared with the corresponding diagonal one. The best micro or macro score for each data set is underlined.

				Softmax Tuning and Testing Multilabel		
				Ekman	Plutchik	POMS
Initial Weight Learning	RNN	Word	Ekman	56.9 / 69.6	37.0 / 47.3	9.8 / 22.1
			Plutchik	52.8 / 67.6	51.0 / 66.5	11.5 / 20.3
			POMS	6.1 / 9.4	3.1 / 4.1	67.0 / 70.4
		Char	Ekman	55.0 / 70.7	40.3 / 54.0	20.3 / 30.4
			Plutchik	55.7 / 70.5	52.8 / 68.3	23.4 / 33.2
			POMS	21.5 / 32.7	12.0 / 16.6	67.5 / 70.5
	CNN	Word	Ekman	57.8 / 69.2	51.0 / 63.6	51.9 / 55.5
			Plutchik	59.3 / 69.5	55.5 / 67.3	50.9 / 55.8
			POMS	44.4 / 57.1	39.6 / 51.6	65.9 / 68.3
		Char	Ekman	59.1 / 70.1	51.5 / 64.2	51.8 / 56.1
			Plutchik	56.4 / 69.7	55.5 / 67.0	51.7 / 56.9
			POMS	49.2 / 61.5	43.4 / 55.9	64.4 / 66.7

to POMS and from POMS to either Ekman or Plutchik. Further, the performance of multilabel models is much worse than that of multiclass models. As models in multilabel mode need to provide a decision for each of the categories, it is reasonable that working on insufficient representation hurts them more than in multiclass mode.

Results for CNNs are a bit more optimistic. In the multiclass mode on characters, the transfer from Ekman to POMS gave the 57.8 % micro-averaged F1-score, while the single model yielded 69.8 %. Comparing to RNNs, the performance loss is much smaller, which we attribute to a simpler architecture of CNNs which is consequently less prone to over-fitting.

Although the performance drops are smaller for CNNs than for RNNs, we conclude

that the final hidden state representations trained on one data set are insufficient for recognising other classifications without significant performance drops.

5.4 Unison Learning

Before presenting the unison results, we turn our attention to accuracy plots of different training heuristics through iterations presented in Figure 5.1. Although we only show plots for character-based RNN, the same trends were observed for other settings as well.

For the alternating batches (AB) strategy we observed the following. The accuracy on the validation set for Ekman and Plutchik was almost flat and stopped increasing in the last 15 epochs. The accuracy on the train set, however, was still rising, making the difference between the train and validation accuracies larger and larger. This indicates that the validation accuracy has probably peaked and that the model would start to overfit if the training continued. Contrary, the validation accuracy for POMS was still rising through all epochs — it had the largest value in the very last epoch of the plot. Almost in parallel with it, the train accuracy was also rising. As the difference between train and validation accuracy was drastically smaller for POMS than for Ekman and Plutchik, and as both train and validation accuracy were still rising for POMS, we concluded that POMS was still in the process of training and did not yet start to overfit. Hence, we believe that validation accuracy for POMS would still increase if the training continued. The reason that training was stopped, despite POMS not yet being trained to its fullest potential, is that the early stopping criterion was met. As we monitored the average validation accuracy across all three data sets, and as it has not improved in the last five epochs, the training was stopped. To sum up our observation about the alternating batches strategy, we believe that the model had enough time to sufficiency train for Ekman and Plutchik, but did not saw enough examples to sufficiently train for POMS. This discrepancy between the number of examples a neural network must observe during training to extract sufficient information from a specific data set was the primary motivation for the development of our novel training heuristic. When observing the plot for weighted sampling batches (WSB), we noticed that the total number of epochs got larger for all eight experimental settings. Observing the differences between train and validation data sets, we noticed that they became smaller for Ekman and Plucthik and larger for POMS — making them much more similar than with AB strategy. This indicates that the network overfit less for Ekman and Plutchik and trained better for POMS.

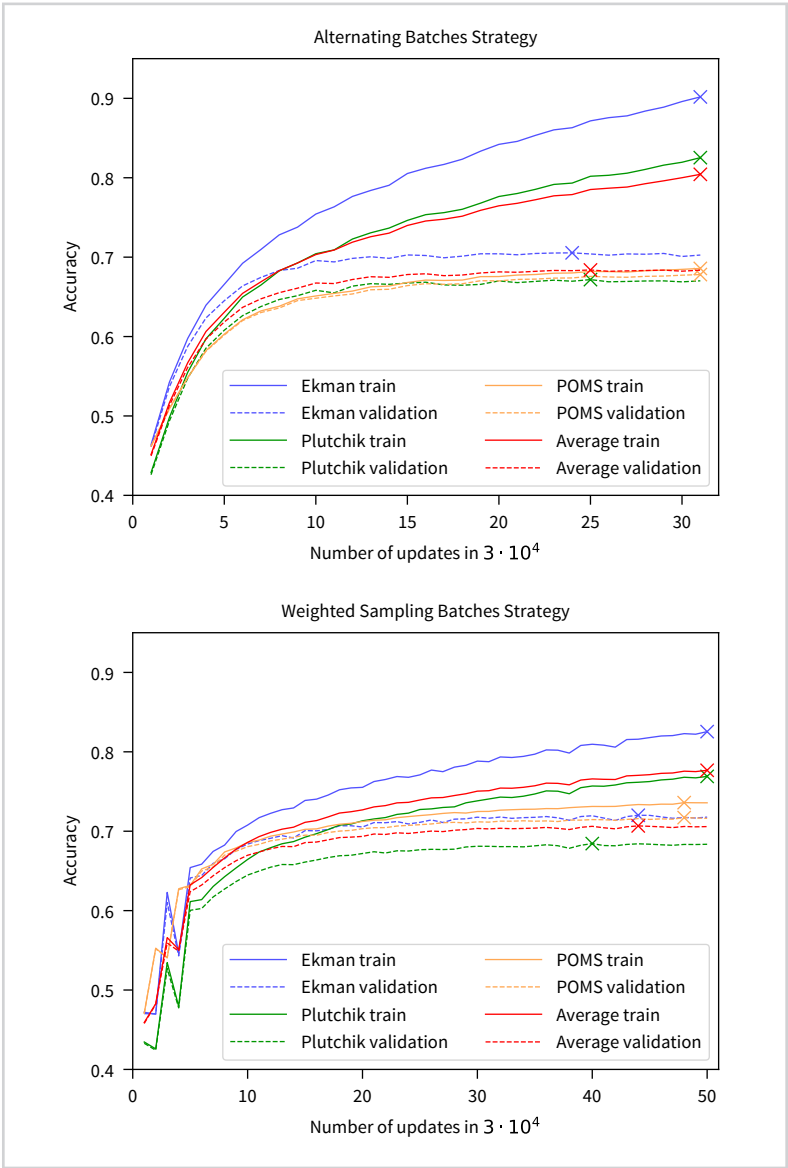


Figure 5.1
Comparison of accuracies through iterations of a unison model training with alternating batches (top) and weighted sampling batches strategies (bottom) in train-validation setting. Cross marks indicate on the maximum of each curve. Sampling probabilities corresponding to the weighted sampling batches strategy are shown in Figure 5.2. We are showing a comparison on characters in the multiclass setting using RNNs while the same trends are present in other settings as well.

Sampling probabilities of each data set through all epochs are presented in Figure 5.2. After some oscillation, sampling probabilities converged to approximate proportions of 0.14, 0.16, 0.70 for Ekman, Plutchik, and POMS correspondingly. The WSB strategy did indeed devote much more attention to POMS; the number of examples coming from POMS was larger than that of Ekman and Plutchik combined. For reference, the weighted sampling batches by data set sizes (WSBDS) used the proportions 0.07, 0.10, 0.83, which would halve the number of training examples for Ekman.

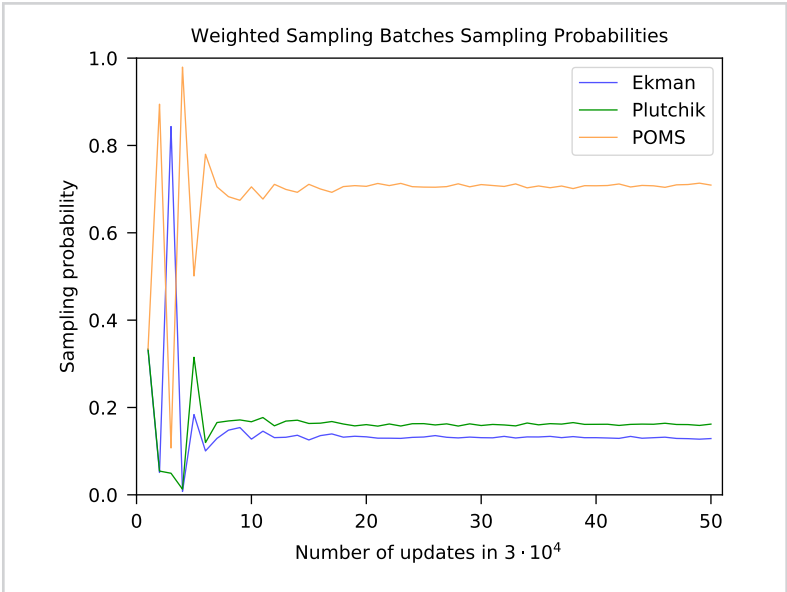


Figure 5.2
Sampling probabilities of weighted sampling batches strategy corresponding to the accuracy plot shown in Figure 5.1. These probabilities were estimated in the multiclass setting using characters as input to RNNs while plots for other settings exhibit the same trends.

In spite of accuracy plots indicating that WSB boosted the performance for POMS, the real assessment of the heuristic is the performance on the test set. For all three training heuristics, it is presented in Table 5.10 for multiclass and in Table 5.11 for the multilabel mode.

First, we observed that for the AB strategy, the performance of Ekman and Plutchik is comparable to the performance of single models from Section 5.2 and is usually within 1 % deviation in absolute. Contrary, the performance for POMS is degraded by up to 5 % in absolute comparing to the single model from Section 5.2. This confirms our obser-

Table 5.10

F1-scores (macro/micro) of multiclass unison model on the test set when trained with all three strategies. In each setting, the higher score between alternating batches (AB), weighted sampling batches (WSB) and weighted sampling batches according to data set sizes (WSBDS) strategy is underlined.

NN	Input	Strategy	Multiclass			
			Ekman	Plutchik	POMS	Harmonic
RNN	Word	AB	61.1 / 71.5	56.7 / 68.5	64.3 / 67.7	60.5 / 69.2
		WSBDS	59.6 / 71.3	56.6 / 68.2	69.1 / 71.7	61.3 / 70.4
		WSB	60.3 / 71.0	57.5 / 68.6	68.2 / 71.1	61.7 / 70.2
	Char	AB	60.4 / 71.9	56.7 / 68.6	63.8 / 68.0	60.2 / 69.5
		WSBDS	58.7 / 71.0	54.6 / 67.8	69.4 / 72.3	60.3 / 70.3
		WSB	61.8 / 73.0	57.8 / 69.5	69.5 / 72.2	62.7 / 71.5
CNN	Word	AB	59.2 / 69.7	54.7 / 66.7	62.9 / 66.2	58.7 / 67.5
		WSBDS	57.6 / 68.2	53.6 / 65.0	66.2 / 69.2	58.7 / 67.4
		WSB	57.8 / 69.1	54.4 / 65.8	65.6 / 68.3	58.9 / 67.7
	Char	AB	60.3 / 71.7	55.2 / 68.1	61.8 / 66.5	59.0 / 68.7
		WSBDS	56.6 / 69.1	52.3 / 65.5	65.3 / 69.0	57.6 / 67.8
		WSB	57.8 / 69.7	52.5 / 66.0	64.8 / 68.5	57.9 / 68.1

vations of accuracy plots: while AB sufficiently trained the unison networks for Ekman and Plutchik, it did not train enough on POMS. Consequently, the performance of unison model for POMS was inferior to that of a single model.

The results for WSB reveal that it was able to bring the performance of POMS up and made it comparable to that of a single model. For example, with RNN working on characters in multiclass mode, the micro-averaged F1-score with AB is 68.0 % while with WSB it is 72.2 %. At the same time, the accuracy for Ekman and Plutchik was not harmed noticeably. Hence, WSB brings us closer to the average performance of three single models while using a single model with shared final hidden state representation instead.

WSB outperformed AB strategy in terms of harmonic mean across all three data sets in the majority of the cases. If we count micro- and macro-averaged scores as separate, WSB outperformed AB in 12 out of 16 cases. Further, WSB outperformed WSBDS in all cases,

Table 5.11

F1-scores (macro/micro) of multilabel unison model on the test set when trained with all three strategies. In each setting, the higher score between alternating batches (AB), weighted sampling batches (WSB) and weighted sampling batches according to data set sizes (WSBDS) strategy is underlined.

NN	Input	Strategy	Multilabel			
			Ekman	Plutchik	POMS	Harmonic
RNN	Word	AB	58.2 / 71.5	54.0 / 68.0	62.2 / 65.9	57.9 / 68.4
		WSBDS	52.8 / 68.5	49.4 / 65.1	67.2 / 69.9	55.5 / 67.8
		WSB	55.1 / 70.1	51.2 / 66.8	65.8 / 69.0	56.7 / 68.6
	Char	AB	56.3 / 71.0	52.7 / 67.3	60.2 / 65.0	56.2 / 67.7
		WSBDS	54.3 / 69.6	49.4 / 65.9	67.8 / 70.9	56.2 / 68.8
		WSB	55.1 / 70.4	52.3 / 67.3	66.2 / 69.7	57.3 / 69.1
CNN	Word	AB	57.2 / 69.2	53.6 / 66.4	62.1 / 65.4	57.4 / 67.0
		WSBDS	53.1 / 66.8	48.0 / 64.0	65.3 / 67.7	54.6 / 66.2
		WSB	56.6 / 68.2	53.3 / 65.0	64.8 / 67.0	57.9 / 66.7
	Char	AB	59.9 / 69.3	53.4 / 66.9	61.6 / 63.4	58.1 / 66.5
		WSBDS	57.5 / 68.7	52.8 / 65.1	65.9 / 67.9	58.2 / 67.2
		WSB	60.4 / 70.1	54.0 / 67.1	65.8 / 67.6	59.7 / 68.3

except for RNN in a multiclass mode with words on input. We must emphasise that in terms of the harmonic mean, the best performing unison model in either multiclass or multilabel setting was achieved by training with WSB method.

This validates that neither simply alternating batches nor sampling according to data set sizes are sufficient for data sets that differ in size or complexity, but an adaptive heuristic able of estimating the progress of training is required. Also, this empirically validates our intuition that the difference between train and validation accuracies can be used as a proxy for estimating the stage of overfitting. Consequently, this difference can be used to guide the sampling heuristic to train a model whose performance across all data sets is more balanced.

5.5 Unison Transfer Learning

In this section, we test the generality of our unison model embedding (i.e. the final hidden states of the neural network) by applying it to other related classification tasks. Recall that in Section 5.3 we showed that embeddings trained on a single models did not reveal to be directly useful for other tasks, at least not without great performance drops. Then in Section 5.4 we trained a unison model and showed that it is sufficient for all of the three data sets used in our study. Hence in this section, we investigate whether the unison model embedding is general enough to be useful for other related classification tasks. In the experiments we use these four data sets:

- *Sentiment 140 [1]:* a data set for analysing sentiment of tweets (positive or negative) containing 1.600.000 training and 359 testing examples. Training data was collected with distant supervision (emoticons) while testing data was manually annotated. We split the training data into two sets: one for training (1.200.000) and one for validation (400.000), and used the already provided testing set for final evaluation.

- *SemEval 2017, Task 4 [81]:* a data set for analysing sentiment of tweets (positive, neutral, or negative) containing 53.570 training and 12.284 testing examples. Both training and testing examples were manually annotated. We split the training data into two sets: one for training (40.177) and one for validation (13.393), and used the already provided testing set for final evaluation.

- *Twitter Emotion Corpus (TEC) [82]:* a data set for recognising Ekman's emotions containing 21.051 labelled tweets. The data was collected between November 2011 and December 2011 and the classes were assigned using distant supervision from hashtags as we did in our research. As it was not yet split, we randomly split it into training (12.630), validation (4.210), and testing (4.211) sets.

- *Political Tweets [83]:* a data set for recognising Plutchik's emotions containing 4.056 labelled tweets from the 2012 US presidential elections. The data was manually annotated using Amazon's Mechanical Turk. As it was not yet split, we randomly split it into training (2.433), validation (811), and testing (812) sets.

Although the tweets for the last two data sets were collected in the same time span as

we collected ours, we checked and found no overlap between TEC or Political Tweets and our data sets.

For each of the above data sets, we trained three neural networks. First one was trained from scratch by randomly initialising all neural network weights. Doing so, we estimated the performance of the neural network on a given data set without any external help. The second neural network was initialised to our unison model embedding while the weights corresponding to this embedding were fixed. Hence, only the final softmax layer was trained, similarly as we did in Section 5.3. This forced the neural network to use the same final hidden state representations as trained with the unison model. This tested whether the unison model embedding can be directly applied to other tasks. Finally, the third neural network was initialised as the second one, but all weights were updated during training (i.e. the weights corresponding to the initialised embedding as well as the final softmax weights). This tested how beneficial is the unison model embedding when used as an initialiser which can be further tuned. Notice that all three neural networks were trained under the same conditions (i.e. the architecture, optimisers, etc.) while the only difference is whether we initialised the weights to our embedding or not and whether we allowed the initialised embedding to be further tuned or not.

Table 5.12

Performance on test sets when applying our embedding to other data sets. The best performance among the three neural networks is underlined. We report the same metrics as the original paper along with the best score from that paper in the last column for comparison.

Data Set	Metric	Weight Initialisation			*Best score from the paper*
		None	Initialised fixed	Initialised fine-tuned	
Sentiment 140	CA	81.3	74.9	<u>84.4</u>	*83.0*
SemEval 2017	Macro Recall	51.6	<u>54.0</u>	47.6	*68.1*
TEC	Micro F1	45.3	<u>64.3</u>	62.1	*49.9*
Political Tweets	CA	40.4	45.6	<u>49.8</u>	*56.8*

Performance of the three such networks on all testing sets is presented in Table 5.12. For comparison, we also report the best performance achieved by the paper that introduced the data set and used the same evaluation metric. We observed that for all four data sets by using either fixed embedding or allowing it to be further fine-tuned, we im-

proved the performance in comparison with the neural network trained from scratch. This indicates that our unison model embedding contains some knowledge, which cannot necessarily be obtained solely from the other task (given its corresponding data) and which can help to improve the performance. For Sentiment 140 data set we surpass the performance reported in the paper with the help of fine-tuning. Contrary, for SemEval 2017 our performance is quite lower in comparison with the best performance achieved in the competition. A potential reason for this is that we did not focus on neutral tweets while they were present in the testing set. For both emotion data sets, the performance of either fixed or fine-tuned neural network is better than the one trained from scratch. We observed the most significant improvement for TEC data set, both comparing to no initialisation as well as to the best performance of the original paper, which is expected as TEC contains the data of the same type as our classifiers were trained on, but we used orders of magnitude larger training data set.

We believe that unison model embedding is a viable candidate for improving the performance on other tasks. Whether it should be used and fixed or whether it should be fine-tuned is the parameter that should be tested as it depends on how closely related the target task is to the task of emotion recognition for which our classifiers were trained.

5.6 Comparison of Emotion Classifications

In this section, we investigate the correlations between classes of different emotion classifications used in this study. We are mainly interested in the mappings between different emotion classifications; i.e. if we know an emotion from the first classification is expressed in the text, can we guess what emotions from the second classification might also be applicable? To investigate this, we designed the following experiment. We took a test data set of one of the emotion classifications (e.g. Ekman) and applied a classifier that recognises emotions from another emotion classification (e.g. POMS). Then we present the mapping of actual labels from the first classification to predicted labels in the second classification in a confusion matrix. In all experiments, we used the best performing classifiers trained with the unison model approach (see Section 5.4).

We considered making these comparisons by applying two different classifiers on the same data set but decided to instead compare actual labels from one classification with the predictions for the others as labels might be more accurate than the classifier's predictions. However, we observed that comparing predictions of two classifiers yielded very

similar results to the ones we present.

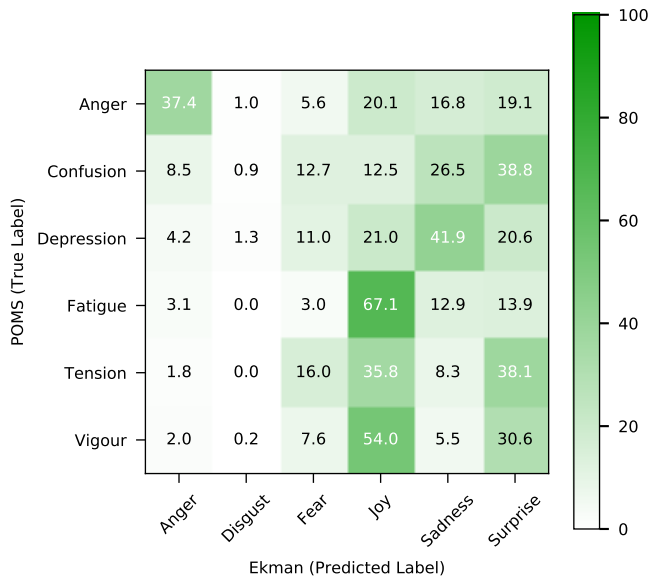


Figure 5.3
Predictions of Ekman's classifier when applied to POMS's data set in multiclass mode. Each cell shows the percentage of corresponding examples; percent signs are omitted due to brevity.

The confusion matrix presenting the predictions of Ekman's classifier on POMS's data is shown in Figure 5.3. We observed that some categories map nicely. *Anger* in most cases maps to *anger*, which is expected as this is essentially the same category in both classifications. Most *depression* examples map to *sadness*, *vigour* to *joy*, and *confusion* to *surprise*, all of which seem related. *Tension* is split between *surprise* and *joy*, while the greater majority of examples expressing *fatigue* map to category *joy*. We suspect the latter is the consequence of not handling sarcastic tweets while we know that there are many sarcastic uses of hashtag *#joy* inside the Ekman's data set. Hence, the classifier learned to put both genuinely joyful as well as sarcastic tweets into the *joy* category. We believe that for the majority of the tweets from the *fatigue* category Ekman's classifier suspected they might be sarcastic and hence used the *joy* category.

Results of applying POMS's classifier on tweets from Plutchik's data set are shown in

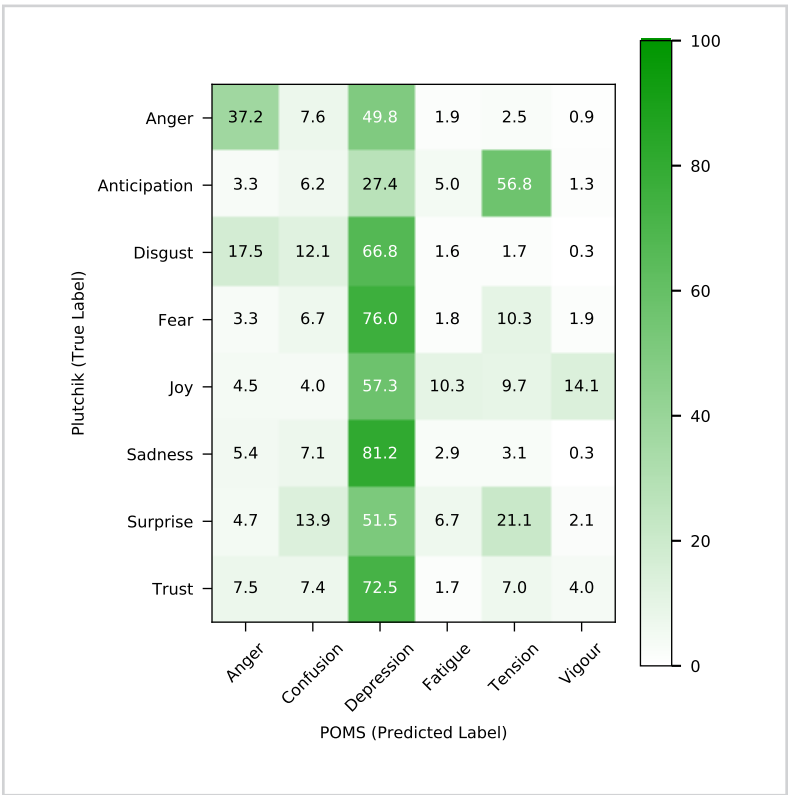


Figure 5.4
Predictions of POMS's classifier when applied to Plutchik's data set in multiclass mode. Each cell shows the percentage of corresponding examples; percent signs are omitted due to brevity.

Figure 5.4. We observe that the trends are less obvious than in the previous case. A lot of *anger* examples again map to *anger* category, while most tweets expressing *anticipation* map to *tension* category. For other categories from Plutchik's classification we notice a significant bias towards the *depression* category in POMS with some mostly negative categories (i.e. *sadness*, *fear*, and *disgust*) mapping more often to *depression* than to some mostly positives ones (i.e. *joy* and *surprise*). We believe this happened due to a combination of two reasons. First, the *depression* is the most common category in POMS's data set; hence the classifier is biased towards it. Second, performing these experiments in multiclass setting forced the classifier to map all of the Plutchik's categories to POMS's,

although for some Plutchik's categories there might not be a suitable candidate in the POMS's classification.

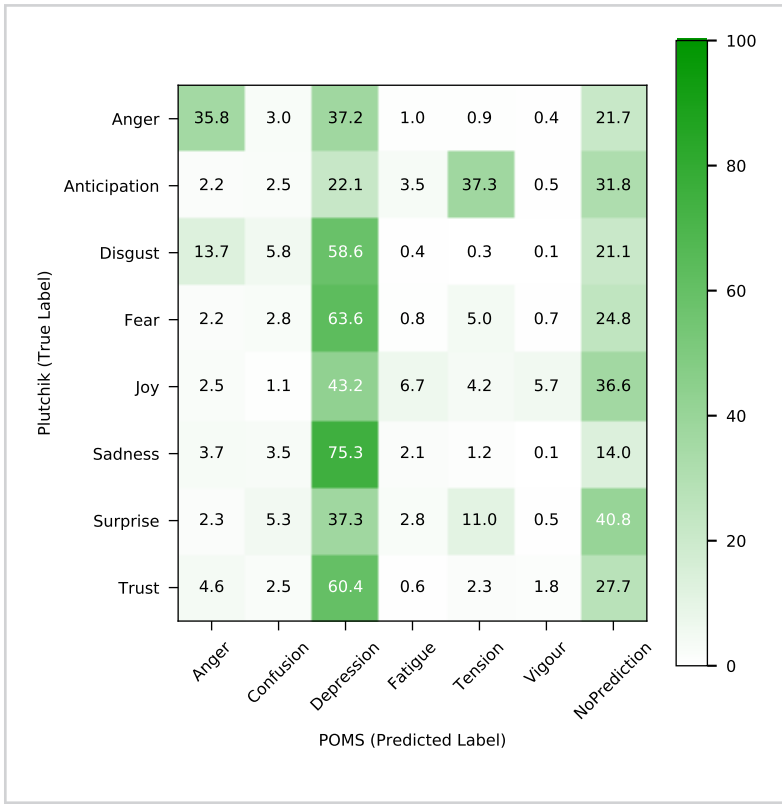


Figure 5.5
Predictions of POMS's classifier when applied to Plutchik's data set in multilabel mode. Cases when the classifier did not recognise any of the emotions are shown in *NoPrediction* column. Each cell shows the percentage of corresponding examples; percent signs are omitted due to brevity.

To allow the classifier not to provide a mapping from the source to the target category we repeated the same experiments in multilabel setting and show the results in Figure 5.5. In comparison to the multiclass mode, we observed that for many examples that were previously mapped to *depression* category the classifier did now not recognise any of POMS's emotions. In Figure 5.5 we mark such cases as the *NoPrediction* category. Especially for mostly positive categories *joy* and *surprise* there are now around 40 % tweets for which the classifier did not assign POMS's category, while in the multiclass

mode it was forced to make a decision.

5.7 Limitations and Future Work

Most of study's limitations stem from the data set creation process. First, we restricted ourselves to exact matches of emotional hashtags. Expanding these sets to include morphological variations of the emotional words as well as their synonyms, could drastically enlarge the number of matching tweets and consequently our data set sizes, while also expanding the variation of tweets inside each category.

The class distributions are quite imbalanced, as we show in Section 3.2. Consequently, some of our classifiers perform quite poorly for categories containing a few examples. The confusion matrix in Figure 5.6 shows that classifiers are biased towards categories with the large number of examples (e.g. *joy*), while they perform quite poorly for categories with very few examples (e.g. *disgust*). This is also confirmed by the Table 5.13, which shows the performance for each of the categories separately. We observed that the performance is directly correlated with the number of examples inside the category. The more examples there are for a given category, the better the model performs on it. For the *disgust* category, with only 1,419 examples, we achieved the F1-score of 23.6 %, while for *joy*, with 36,958 examples, we achieved the F1-score of 78.0 %. Although we only report the numbers for Ekman here, the same trends are present for Plutchik and POMS as well. We could devote more effort to make data sets more balanced either in the data set creation process, by including synonyms as mentioned above, or later by under- or oversampling. An interesting idea for balancing our sets would be to try generating new training examples, like in visual recognition competitions by applying transformations to images. We could replace some words in the tweet's content with synonyms to obtain a *new* training example corresponding to the same category. Repeating these substitutions for the categories with few training examples could help us balance the class distributions, but whether this would improve the performance for categories with few examples is unknown.

Our data sets only contain tweets corresponding to the emotional categories but no neutral tweets, mainly as finding them with distant supervision approach is not easy. Although the multilabel setting supports the neutral class (i.e. when the classifier returns negative prediction for all categories), we neither trained them on such examples nor tested their performance on neutral tweets. Hence, we know that our models can

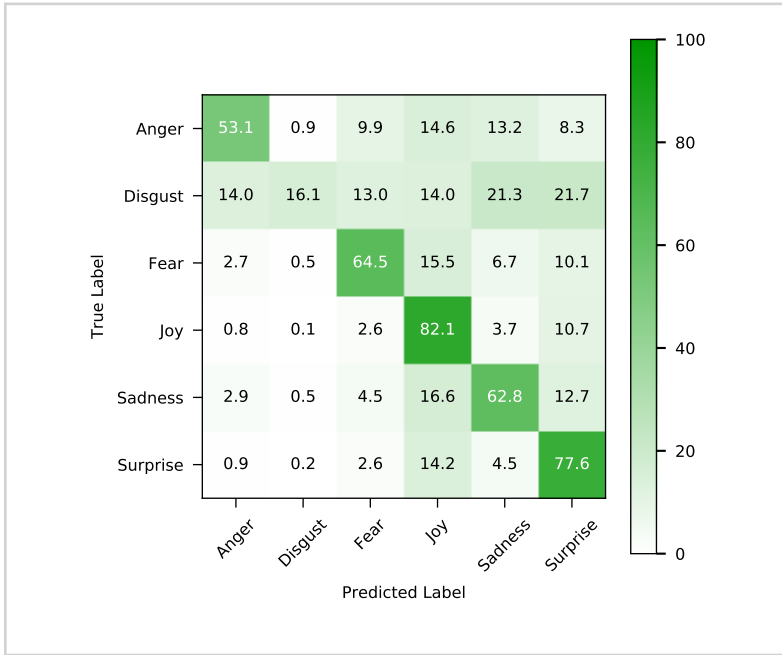


Figure 5.6
Confusion matrix on the Ekman's test data for unison model trained using weighted sampling batches strategy. Each cell shows the percentage of corresponding examples; percent signs are omitted due to brevity.

distinguish between fine-grained emotional states, but how they perform on a stream of tweets containing neutral examples is an interesting question for future work.

When creating the data sets by distant supervision instead of using human annotators, some tweets can be mislabelled. We observed that category *joy* contains some sarcastic tweets such as, for example, *"I feel like crap! An elephant is sitting on my chest and my stomach is churning! #joy #exhausted #wornout"*. Such sarcastic tweets, as well as other mislabelled examples, are introducing erroneous training instances, which confuses the classifiers while training or misleads them into recognising incorrect patterns. While they might not be problematic from a performance evaluation perspective, as the classifiers can learn to put both joyful as well as sarcastic tweets into the same category, they are very problematic in terms of results interpretation. Recall that we spotted this in the previous section where the Ekman's classifier predicted the category joy for tweets actually expressing fatigue. As sarcasm is a hard problem even in texts longer than tweets and

Table 5.13

Performances on single categories evaluated on Ekman's test data for the unison model trained using weighted sampling batches strategy. For each emotion, we report the precision, recall, F1-score, and support (i.e. the number of tweets belonging to that category).

	Precision	Recall	F1-Score	Support
Anger	66.6	53.1	59.1	6,134
Disgust	44.6	16.1	23.6	1,419
Fear	74.8	64.6	69.3	14,970
Joy	74.2	82.2	78.0	36,958
Sadness	66.9	62.9	64.8	15,820
Surprise	74.9	77.7	76.3	31,843

is sometimes impossible to deduct from the content itself without modelling authors, we did not tackle it in our study.

Further, while we showed that the proposed WSB heuristic works in the domain of emotion recognition, it would be interesting to test it on different domains inside and outside of the field of NLP.

Finally, before putting out classifiers into the wild, their performance should be tested on a set of unfiltered, preferably human-annotated set of tweets. Even if the training data remains labelled by distant supervision, evaluation on a data set labelled by human annotators would give valuable information regarding the performance we can expect in the wild.

6

Showcases

We made our best performing models, i.e. RNN models on characters in both multiclass and multilabel mode, publicly available. First, they are accessible through the Python programming language, which we present in Section 6.1. Secondly, they are available in Orange [84], an open source machine learning and data visualisation toolbox, which we present in Section 6.2. As Orange is based on visual programming and does not require the knowledge of coding, this makes our models accessible also to people unskilled in programming. Further, this supports future research to compare with our work on new data sets easily.

6.1 Python

The package *twitter-emotion-recognition* for Python 3 is freely available on GitHub at https://github.com/nikicc/twitter-emotion-recognition. To install it, clone the repository and install its dependencies defined in *requirements.txt* file. We illustrate its functionality on a simple example. First, import the EmotionPredictor class, which serves as the interface to our models.

```
>>> from emotion_predictor import EmotionPredictor
>>> model = EmotionPredictor(classification='ekman', setting='mc')
```

To select the emotion classification, set the *classification* argument to *ekman*, *plutchik*, or *poms*. To switch from multiclass to multilabel mode, set the *setting* parameter to *ml*. With the model initialised, we define a list of tweets that we use to illustrate the functionalities of the model.

```
>>> tweets = [
    "Watching the sopranos again from start to finish!",
    "Finding out i have to go to the dentist tomorrow",
    "I want to go outside and chalk but I have no chalk",
    "I HATE PAPERS AH #AH #HATE",
    "My mom wasn't mad",
    "Do people have no Respect for themselves or others peoples homes",
]
```

To recognise emotions, we use the *predict_classes* method. It returns a Pandas [85] data frame with two columns. The *Tweet* column contains the tweet content, while the *Emotion* column contains the emotion our model recognised in that tweet.

```
>>> model.predict_classes(tweets)
                                                          Tweet   Emotion
0              ”Watching the sopranos again from start to finish!”       Joy
1               ”Finding out i have to go to the dentist tomorrow”      Fear
2            ”I want to go outside and chalk but I have no chalk”   Sadness
3                                    ”I HATE PAPERS AH #AH #HATE”     Anger
4                                              ”My mom wasn’t mad”  Surprise
5  ”Do people have no Respect for themselves or others peoples...”   Disgust
```

To obtain probabilities for each of the emotions, we use the method *predict_probabilities*. Classifiers in the multiclass mode return the probability distribution across all categories, while in the multilabel mode, they return independent probabilities for each of the categories.

```
>>> model.predict_probabilities(tweets)
                Tweet    Anger  Disgust     Fear      Joy  Sadness  Surprise
0  ”Watching the ...”  0.00072  0.00024  0.00383  0.94654  0.00561   0.04306
1  ”Finding out i...”  0.00770  0.00004  0.78389  0.19863  0.00895   0.00079
2  ”I want to go ...”  0.00277  0.00010  0.00414  0.02503  0.96371   0.00425
3  ”I HATE PAPERS...”  0.95634  0.00637  0.03139  0.00035  0.00437   0.00118
4  ”My mom wasn’t...”  0.06397  0.00499  0.01397  0.07988  0.21871   0.61848
5  ”Do people hav...”  0.09899  0.79255  0.05698  0.00290  0.03184   0.01675
```

Finally, besides emotion recognition, we also support embedding the content of tweets into fixed length vectors that can be used for any further analysis, such as training custom classifiers or performing unsupervised machine learning (e.g. clustering). This fixed length vector representation corresponds to the final hidden state of the recurrent unison model. To embed the tweets, use the *embed* method. It returns the representation of the tweets as vectors of length that is equal to the dimensionality of the final hidden state representation of the model (i.e. 800 in the below case).

```
>>> model.embed(tweets)
                Tweet     Dim1     Dim2  ...     Dim798   Dim799   Dim800
0  ”Watching the ...” -0.12876 -0.00000  ...    -0.26090 -0.00906 -0.11021
1  ”Finding out i...” -0.52560  0.40785  ...    -0.00009 -0.00149  0.14287
2  ”I want to go ...” -0.05785  0.56642  ...    -0.09134 -0.00391 -0.03748
3  ”I HATE PAPERS...”  0.01967 -0.28851  ...     0.10023  0.01335 -0.01430
4  ”My mom wasn’t...” -0.00414  0.65758  ...    -0.02932 -0.00746 -0.06621
5  ”Do people hav...” -0.20141  0.08131  ...     0.03366  0.01274 -0.00501
```

6.2 Orange

After installing Orange (https://orange.biolab.si/) we install the Orange3-Text add-on, which includes components for textual analysis as well as the *Tweet Profiler* widget with our models. In Orange's menu select *Options -> Add-ons* to open the add-on window and make sure *Text* add-on is installed, as shown in Figure 6.1.

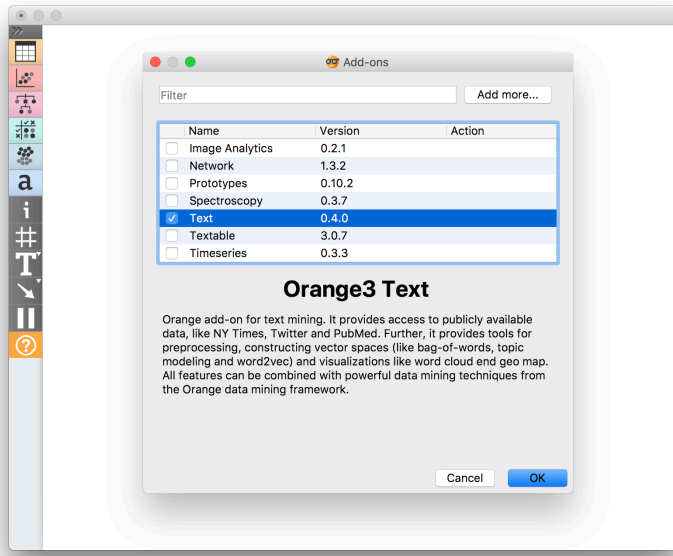


Figure 6.1
Installing Orange3-Text add-on which contains the *Tweet Profiler* widget with our models. The add-on dialog is under *Options -> Add-ons*.

With the Text addon installed, we can start using the emotion-recognition models. First, let us load some data. For this showcase, we use the data set of tweets by Hillary Clinton and Donald Trump in the time before the 2016 elections, which comes along with the addon. To open the data set, place the *Corpus* widget on the canvas and open it. Then click on the *Browse documentation corpora* button and pick *election-tweets-2016.tab* as shown in Figure 6.2. We see that the corpus contains 6444 tweets and has a discrete class with two values stating whether Donald or Hillary was the author of the tweet.

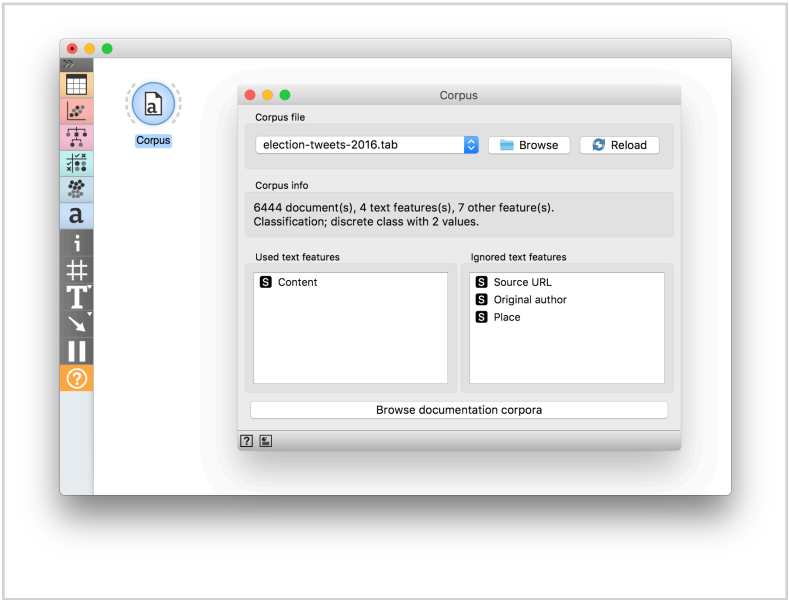
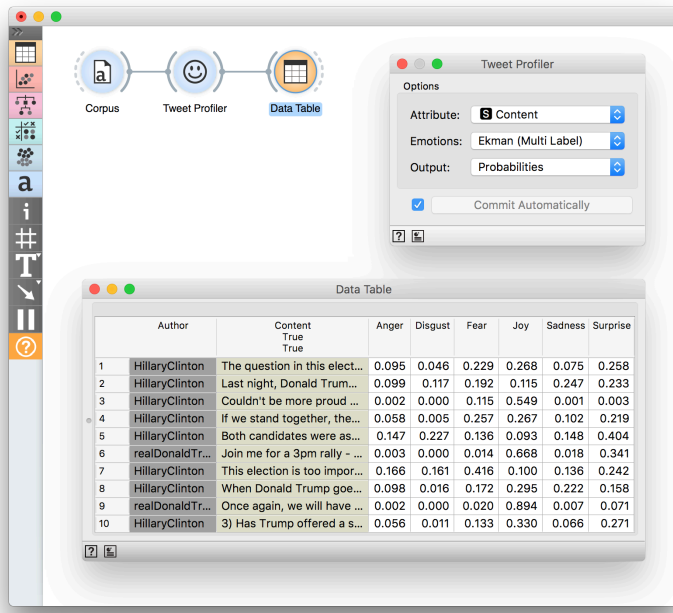


Figure 6.2
Loading the *election-tweets-2016* data set with the Corpus widget.

With the data loaded, we now pass it to *Tweet Profiler* widget, which serves as the interface to our models. The widget sends the data to a remote server where the models are run and then the results are sent back to Orange for further processing. The Tweet Profiles widget, as shown in Figure 6.3, has several options. First, in the *Attribute* setting we define what should be the input to the emotion recognition models. In this example, we use the *Content* feature, as this is where the content of the tweet is stored. Next, we select the emotion classification and the classifier output mode (i.e. multiclass or multilabel) in the *Emotions* option. Finally, the *Output* option defines the output we want from the model. Selecting *Classes* returns the emotion classes as predicted by the model, selecting *Probabilities* returns the probabilities for the emotion classes, while *Embeddings* embeds the tweets and returns their final hidden state representations.

The output of the Tweet Profiler widget now contains the data enhanced with six additional features. For each of the Ekman's emotional categories — anger, disgust, fear, joy, sadness, and surprise — the model added the probability for it to be expressed in the tweet. The probabilities can be observed by passing the output of Tweet Profiler widget



	Author	Content True True	Anger	Disgust	Fear	Joy	Sadness	Surprise
1	HillaryClinton	The question in this elect...	0.095	0.046	0.229	0.268	0.075	0.258
2	HillaryClinton	Last night, Donald Trum...	0.099	0.117	0.192	0.115	0.247	0.233
3	HillaryClinton	Couldn't be more proud ...	0.002	0.000	0.115	0.549	0.001	0.003
4	HillaryClinton	If we stand together, the...	0.058	0.005	0.257	0.267	0.102	0.219
5	HillaryClinton	Both candidates were as...	0.147	0.227	0.136	0.093	0.148	0.404
6	realDonaldTr...	Join me for a 3pm rally - ...	0.003	0.000	0.014	0.668	0.018	0.341
7	HillaryClinton	This election is too impor...	0.166	0.161	0.416	0.100	0.136	0.242
8	HillaryClinton	When Donald Trump goe...	0.098	0.016	0.172	0.295	0.222	0.158
9	realDonaldTr...	Once again, we will have ...	0.002	0.000	0.020	0.894	0.007	0.071
10	HillaryClinton	3) Has Trump offered a s...	0.056	0.011	0.133	0.330	0.066	0.271

Figure 6.3
Tweet Profiler widget sends the data to a remote server where our models are run. The results, the probabilities for Ekman's emotions in this case, are then sent back to Orange and are available on the widget's output channel. Passing the output to *Data Table* widget shows the six new emotion features that the model appended to the data set.

to the *Data Table* widget.

Although observing the probabilities directly is interesting by itself, we can also use them to search for the tweets that strongly express certain emotions. To do so, pass the data to the *Select Rows* widget, as shown in Figure 6.4. In the *Conditions* area, select the *Sadness* feature, set the second dropdown to *is greater than* and the last field to 0.95. Doing so, the widget finds tweets for which the classifier set the probability for expressing sadness to greater than 95 %. The matching tweets are present on the output of the Select Rows widget.

Finally, we pass the output of the Select Rows widget into the *Corpus Viewer* widget, which is more appropriate for presenting textual data than Data Table. In Corpus Viewer, under the *Display features* we select only *Sadness* and *Content*, which hides all

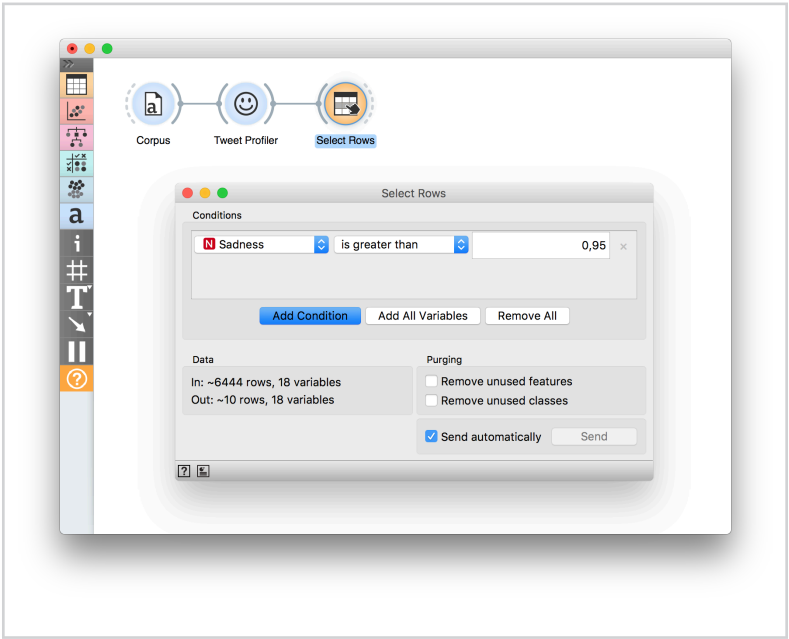


Figure 6.4
Select Rows widget is used to select tweets with the probability of expressing sadness greater than 95 %.

other features. Selecting multiple documents shows them one below the other, as seen in Figure 6.5. The content of the tweets along with the probabilities of expressing sadness as predicted by the classifier are now shown in the area on the right.

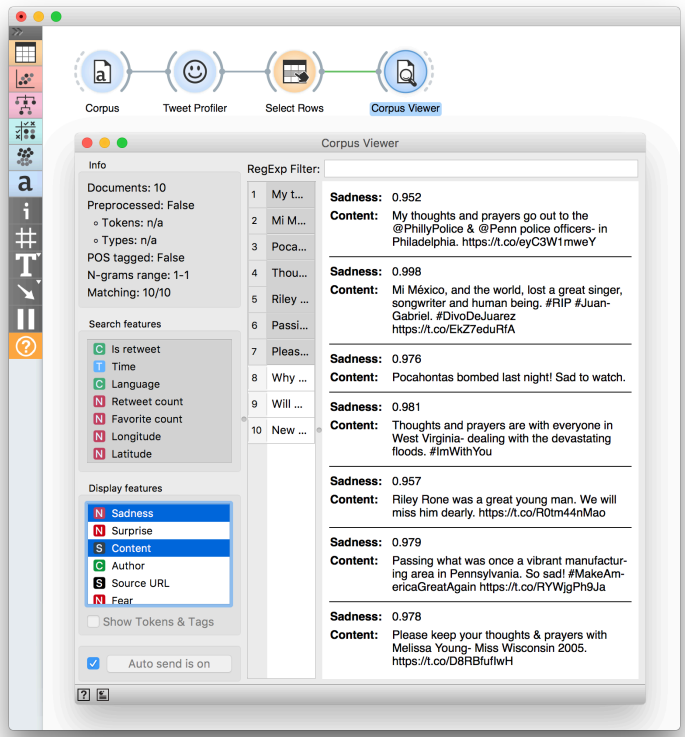


Figure 6.5
We use Corpus Viewer widget to show the tweets to which classifier assigned high probability for expressing sadness.

Conclusion

7

The thesis focuses on emotion recognition on Twitter and is structured to answer two main research questions. First, can we improve the accuracy of emotion-recognition algorithms with the use of neural networks? Deep learning has recently shown impressive improvements in many other areas but has not yet been tested for emotion recognition. And second, can we develop a single model able to recognise multiple emotion categorisations from a shared representation?

To that aim, we first created labelled data sets for Ekman's, Plutchik's, and POMS's emotions by exploiting emotional hashtags as our labels. We then designed a set of experiments that set the baseline performance on these three data sets using traditional approaches to emotion recognition used in the recent literature. When comparing the results of the neural networks with those of the baseline models, we observed that neural networks always show slightly better results than baseline approaches. We discovered that both recurrent and convolution neural networks can perform well; recurrent networks seem a bit more accurate. As the recurrent networks were much more complex than the convolutional ones, we believe the latter to be a valid alternative when less training data is available. All our networks were trained with either words or characters on input. We observed that characters yield better performance if enough training data is available, and that character approach works using both types of networks: recurrent and convolutional. Whenever sufficient data is available, we encourage the use of character approach instead of working with words. There are several benefits to the character approach. It does not require a tokeniser, any token normalisation (like stemming or lemmatisation) or any other language preprocessing tools. It truly is a completely end-to-end learning approach, and as such, it is very easily transferable to other languages, which is another benefit of using character-based neural networks in comparison with traditional methods. As a first study which was recognising POMS's categories from the textual content, we believe they are as predictable as Ekman's and Plutchik's. Also, grouping its adjectives into a smaller set of categories yields a coherent data set suitable for training emotion-recognition classifiers.

Experiments with our neural networks revealed that the transfer capabilities of our models trained on single data sets are poor. This indicates that the discovered final hidden state representations were too specific to be useful for recognising other emotion categorisations. To improve the generality of these hidden state representations, we designed a single model for recognising all three emotion classifications. When such a model was trained with the existing approach, we observed a drastic drop in perfor-

mance for one of our data sets — POMS. As this data set is about ten times larger than the other two, the networks trained sufficiently for the first two data sets but undertrained for POMS. To train a more balanced unison model, we proposed a novel training heuristic for training unison models. Its central idea is that the differences in data set sizes or complexities should be considered when training a model. Hence, instead of sampling training instances uniformly from all data sets, it samples examples based on progress estimates for each of the data sets. We proposed a progress estimation metric based on observing the difference between train and validation accuracy and confirmed its usefulness for guiding the training process. We showed that the newly proposed training heuristic outperformed the existing approach with respect to the harmonic mean of performance across all data sets and hence trained a much more balanced model. The performance of so-trained unison model on each data set is comparable to the performance of single models, while the unison model was restricted to use the shared representation. Hence, we believe, this representation is more general than those of individual models, which was confirmed by testing the utility of the embedding when applied to sentiment analysis, and can be considered as a general emotion representation of the input tweet.

Comparing to previous research, we worked on probably the largest emotion labelled data sets spanning just under seven years and did not restrict ourselves to a particular domain (e.g. finance) but rather tested the usefulness of our models across all available tweets. We conducted all experiments in both multiclass as well as multilabel modes, while we compared word- and character-based inputs for both convolutional and recurrent neural networks. Also, we worked with three emotion classifications at the same time, which not only enabled their comparison on the same type of data but also allowed us to exploit the benefits of multi-task learning when developing the unison model. Since the data was annotated automatically and as the best performing model was an end-to-end character based model, our approach is language independent and could easily be adapted to other languages. All our best performing models are publicly available in Python and Orange, which supports easy comparison of future research with our models. Among possible thesis improvements, the most promising ones are handling of sarcastic and neutral tweets, resolving class imbalance issues, and evaluating models in the wild.

We showed that neural networks should be preferred over traditional text classification approaches for emotion recognition and that, with the newly proposed training heuristic, we can train a single model able of recognising multiple emotion categorisa-

tions from a shared representation.

BIBLIOGRAPHY

[1] Go A, Bhayani R & Huang L (2009) Twitter Sentiment Classification using Distant Supervision. CS224N Project Report, Stanford.

[2] Gruhl D, Guha R, Kumar R, Novak J & Tomkins A (2005) The Predictive Power of Online Chatter. In *Proc. of the 11th ACM SIGKDD Int. Conf. on Knowledge Discovery in Data Mining*. pp. 78–87.

[3] Liu Y, Huang X, An A & Yu X (2007) ARSA: A Sentiment-Aware Model for Predicting Sales Performance Using Blogs. In *Proc. of the 30th Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*. pp. 607–614.

[4] Mishne G & Glance N (2005) Predicting Movie Sales from Blogger Sentiment. In *AAAI Spring Symposium: Computational Approaches to Analyzing Weblogs*. pp. 155–158.

[5] Asur S & Huberman BA (2010) Predicting the Future with Social Media. In *Proc. of the Int. Conf. on Web Intelligence and Intelligent Agent Technology*. pp. 492–499.

[6] Bollen J, Mao H & Zeng XJ (2011) Twitter mood predicts the stock market. *J of Computational Science* 2: 1–8.

[7] Dergiades T (2012) Do investors' sentiment dynamics affect stock returns? Evidence from the US economy. *Economics Letters* 116: 404–407.

[8] Smailović J, Grčar M, Lavrač N & Žnidaršič M (2013) Predictive Sentiment Analysis of Tweets: A Stock Market Application. In *Proc. of the Int. Workshop on Human-Computer Interaction and Knowledge Discovery in Complex, Unstructured, Big Data*. pp. 77–88.

[9] Collobert R et al. (2011) Natural Language Processing (almost) from Scratch. *J of Machine Learning Research* 12: 2493–2537.

[10] Ekman P (1992) An Argument for Basic Emotions. *Cognition and Emotion* 6: 169–200.

[11] Plutchik R (1980) A General Psychoevolutionary Theory of Emotion. In *Theories of Emotion* (Academic Press), pp. 3–33.

[12] McNair DM (1971) *Manual Profile of Mood States* (Educational & Industrial testing service).

[13] Norcross JC, Guadagnoli E & Prochaska JO (1984) Factor Structure of the Profile of Mood States (POMS): Two Partial Replications. *J of Clinical Psychology* 40: 1270–1277.

[14] Bradford RB (2008) An Empirical Study of Required Dimensionality for Large-scale Latent Semantic Indexing Applications. In *Proc. of the 17th ACM Conf. on Information and Knowledge Management*. pp. 153–162.

[15] Mikolov T, Corrado G, Chen K & Dean J (2013) Efficient Estimation of Word Representations in Vector Space. In *Proc. of the Int. Conf. on Learning Representations*. pp. 1–12.

[16] Pennington J, Socher R & Manning CD (2014) GloVe: Global Vectors for Word Representation. In *Proc. of the Conf. on Empirical Methods in Natural Language Processing*. pp. 1532–1543.

[17] Peters ME et al. (2018) Deep contextualized word representations. In *Proc. of the Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. pp. 2227–2237.

[18] Hochreiter S & Schmidhuber J (1997) Long Short-Term Memory. *Neural Computation* 9: 1735–1780.

[19] Cho K et al. (2014) Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. *arXiv preprint* arXiv: 1406.1078.

[20] Chung J, Gulcehre C, Cho K & Bengio Y (2014) Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv preprint* arXiv: 1412.3555.

[21] Kim Y (2014) Convolutional Neural Networks for Sentence Classification. In *Proc. of the Conf. on Empirical Methods in Natural Language Processing*. pp. 1746–1751.

[22] Zhang X, Zhao J & LeCun Y (2015) Character-level Convolutional Networks for Text Classification. In *Advances in Neural Information Processing Systems*. pp. 649–657.

[23] Kalchbrenner N, Grefenstette E & Blunsom P (2014) A Convolutional Neural Network for Modelling Sentences. In *Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics*. pp. 655–665.

[24] Zhang Y & Wallace BC (2016) A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. In *Proc. of the 8th Int. Joint Conf. on Natural Language Processing*. pp. 253–263.

[25] Tieleman T & Hinton G (2012) Lecture 6.5 — RmsProp: Divide the Gradient by a Running Average of its Recent Magnitude. Coursera: Neural Networks for Machine Learning.

[26] Duchi J, Hazan E & Singer Y (2011) Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J of Machine Learning Research* 12: 2121–2159.

[27] Zeiler MD (2012) ADADELTA: An Adaptive Learning Rate Method. *arXiv preprint* arXiv: 1212.5701.

[28] Kingma DP & Ba J (2014) Adam: A Method for Stochastic Optimization. *arXiv preprint* arXiv: 1412.6980.

[29] Caruana R (1997) Multitask Learning. *Machine Learning* 28: 41–75.

[30] Collobert R & Weston J (2008) A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *Proc. of the 25th Int. Conf. on Machine Learning*. pp. 160–167.

[31] Alm CO, Roth D & Sproat R (2005) Emotions from text: machine learning for text-based emotion prediction. In *Proc. of the Conf. on Human Language Technology and Empirical Methods in Natural Language Processing*. pp. 579–586.

[32] Aman S & Szpakowicz S (2007) Identifying Expressions of Emotion in Text. In *Proc. of the Int. Conf. on Text, Speech and Dialogue*. pp. 196–205.

[33] Strapparava C & Mihalcea R (2007) SemEval-2007 Task 14: Affective Text. In *Proc. of the 4th Int. Workshop on Semantic Evaluations*. pp. 70–74.

[34] Strapparava C & Mihalcea R (2008) Learning to Identify Emotions in Text. In *Proc. of the ACM Symposium on Applied Computing*. pp. 1556–1560.

[35] Chaffar S & Inkpen D (2011) Using a Heterogeneous Dataset for Emotion Analysis in Text. In *Canadian Conf. on Artificial Intelligence*. pp. 62–67.

[36] Mohammad SM & Kiritchenko S (2015) Using Hashtags to Capture Fine Emotion Categories from Tweets. *Computational Intelligence* 31: 301–326.

[37] Mohammad SM & Turney PD (2010) Emotions Evoked by Common Words and Phrases: Using Mechanical Turk to Create an Emotion Lexicon. In *Proc. of the NAACL HLT Workshop on Computational Approaches to Analysis and Generation of Emotion in Text*. pp. 26–34.

[38] Mohammad SM, Zhu X, Kiritchenko S & Martin J (2015) Sentiment, emotion, purpose, and style in electoral tweets. *Information Processing and Management* 51: 480–499.

[39] Tromp E & Pechenizkiy M (2014) Rule-based Emotion Detection on Social Media: Putting Tweets on Plutchik's Wheel. *arXiv preprint* arXiv: 1412.4682.

[40] Bollen J & Mao H (2011) Twitter Mood as a Stock Market Predictor. *Computer* 44: 91–94.

[41] Bollen J, Mao H & Pepe A (2011) Modeling Public Mood and Emotion: Twitter Sentiment and Socio-Economic Phenomena. In *Proc. of the 5th Int. AAAI Conf. on Weblogs and Social Media Modeling*. pp. 450–453.

[42] Pepe A & Bollen J (2008) Between Conjecture and Memento: Shaping a Collective Emotional Perception of the Future. In *AAAI Spring Symposium: Emotion, Personality, and Social Behavior*. pp. 111–116.

[43] Neviarouskaya A, Prendinger H & Ishizuka M (2009) Compositionality Principle in Recognition of Fine-Grained Emotions from Text. In *Proc. of the 3rd Int. AAAI Conf. on Web and Social Media*. pp. 278–281.

[44] Neviarouskaya A, Prendinger H & Ishizuka M (2010) @AM: Textual Attitude Analysis Model. In *Proc. of the NAACL HLT Workshop on Computational Approaches to Analysis and Generation of Emotion in Text*. pp. 80–88.

[45] Mishne G (2005) Experiments with Mood Classification in Blog Posts. In *Proc. of the ACM SIGIR Workshop on Stylistic Analysis of Text for Information Access*. pp. 321–327.

[46] Mihalcea R & Liu H (2006) A Corpus-based Approach to Finding Happiness. In *AAAI Spring Symposium: Computational Approaches to Analyzing Weblogs*. pp. 139–144.

[47] Yerva SR, Hoyoung J & Aberer K (2012) Cloud based Social and Sensor Data Fusion. In *Proc. of the 15th Int. Conf. on Information Fusion*. pp. 2494–2501.

[48] Weston J, Chopra S & Adams K (2014) #TagSpace: Semantic Embeddings from Hashtags. In *Proc. of the Conf. on Empirical Methods in Natural Language Processing*. pp. 1822–1827.

[49] Kywe SM, Hoang TA, Lim EP & Zhu F (2012) On Recommending Hashtags in Twitter Networks. In *Proc. of the Int. Conf. on Social Informatics*. pp. 337–350.

[50] Godin F, Slavkovikj V, De Neve W, Schrauwen B & Van De Walle R (2013) Using Topic Models for Twitter Hashtag Recommendation. In *Proc. of the 22nd Int. Conf. on World Wide Web*. pp. 593–596.

[51] Maas AL et al. (2011) Learning Word Vectors for Sentiment Analysis. In *Proc. of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. pp. 142–150.

[52] Dong L et al. (2014) Adaptive Recursive Neural Network for Target-dependent Twitter Sentiment Classification. In *Proc. of the 52nd Annual Meeting of the Association for Computational Linguistic*. pp. 49–54.

[53] Socher R et al. (2013) Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proc. of the Conf. on Empirical Methods in Natural Language Processing*. pp. 1631–1642.

[54] Tai KS, Socher R & Manning CD (2015) Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. *arXiv preprint* arXiv: 1503.00075.

[55] Santos CND & Gatti M (2014) Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts. In *Proc. of the 25th Int. Conf. on Computational Linguistics*. pp. 69–78.

[56] Severyn A & Moschitti A (2015) Twitter Sentiment Analysis with Deep Neural Networks. In *Proc. of the 38th Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*. pp. 959–962.

[57] Ghiassi M, Skinner J & Zimbra D (2013) Twitter brand sentiment analysis: A hybrid system using n-gram analysis and dynamic artificial neural network. *Expert Systems With Applications* 40: 6266–6282.

[58] Arkhipenko K et al. (2016) Comparison of Neural Network Architectures for Sentiment Analysis of Russian Tweets. In *Proc. of the Int. Conf. Dialogue*. pp. 68–76.

[59] Nejat B, Carenini G & Ng R (2017) Exploring Joint Neural Model for Sentence Level Discourse Parsing and Sentiment Analysis. In *Proc. of the 18th Annual SIGDIAL Meeting on Discourse and Dialogue*. pp. 289–298.

[60] Wang X, Jiang W & Luo Z (2016) Combination of Convolutional and Recurrent Neural Network for Sentiment Analysis of Short Texts. In *Proc. of the 26th Int. Conf. on Computational Linguistics*. pp. 2428–2437.

[61] Radford A, Jozefowicz R & Sutskever I (2017) Learning to Generate Reviews and Discovering Sentiment. *arXiv preprint* arXiv: 1704.01444.

[62] Razavian AS, Azizpour H, Sullivan J & Carlsson S (2014) CNN Features off-the-shelf: an Astounding Baseline for Recognition. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition Workshops*. pp. 806–813.

[63] Russakovsky O et al. (2015) ImageNet Large Scale Visual Recognition Challenge. *Int J of Computer Vision* 115: 211–252.

[64] Krizhevsky A & Hinton GE (2012) ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*. pp. 1097–1105.

[65] Szegedy C et al. (2014) Going Deeper with Convolutions. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*. pp. 1–9.

[66] Yosinski J, Clune J, Bengio Y & Lipson H (2014) How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*. pp. 3320–3328.

[67] Nodarakis N, Sioutas S, Tsakalidis A & Tzimas G (2016) Using Hadoop for Large Scale Analysis on Twitter: A Technical Report. *arXiv preprint* arXiv: 1602.01248.

[68] Kouloumpis E, Wilson T & Moore J (2011) Twitter Sentiment Analysis: The Good the Bad and the OMG! In *Proc. of the 5th Int. AAAI Conf. on Weblogs and Social Media*. pp. 538–541.

[69] González-Ibáñez R, Muresan S & Wacholder N (2011) Identifying Sarcasm in Twitter: A Closer Look. In *Proc. of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. pp. 581–586.

[70] Bamman D & Smith NA (2015) Contextualized Sarcasm Detection on Twitter. In *Proc. of the 9th Int. AAAI Conf. on Web and Social Media*. pp. 574–577.

[71] Plank B & Hovy D (2015) Personality Traits on Twitter —or— How to Get 1,500 Personality Tests in a Week. In *Proc. of the 6th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*. pp. 92–98.

[72] Gimpel K et al. (2011) Part-of-Speech Tagging for Twitter: Annotation, Features, and Experiments. In *Proc. of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. pp. 42–47.

[73] Jolliffe IT (2002) *Principal Component Analysis* (Springer).

[74] Rehurek R & Sojka P (2010) Software framework for topic modelling with large corpora. In *Proc. of the LREC Workshop on New Challenges for NLP Frameworks*. pp. 45–50.

[75] Pedregosa F et al. (2012) Scikit-learn: Machine Learning in Python. *J of Machine Learning Research* 12: 2825–2830.

[76] Srivastava N, Hinton G, Krizhevsky A, Sutskever I & Salakhutdinov R (2014) Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J of Machine Learning Research* 15: 1929–1958.

[77] Zaremba W, Sutskever I & Vinyals O (2014) Recurrent Neural Network Regularization. *arXiv preprint* arXiv: 1409.2329.

[78] Chollet F (2015) Keras. URL: https://keras.io.

[79] Al-Rfou R et al. (2016) Theano: A Python framework for fast computation of mathematical expressions. *arXiv preprint* arXiv: 1605.02688.

[80] Devlin J, Chang MW, Lee K & Toutanova K (2018) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint* arXiv: 1810.04805.

[81] Rosenthal S, Farra N & Nakov P (2017) SemEval-2017 Task 4: Sentiment Analysis in Twitter. In *Proc. of the 11th Int. Workshop on Semantic Evaluation*. pp. 502–518.

[82] Mohammad SM (2012) #Emotional Tweets. In *Proc. of the 6th Int. Workshop on Semantic Evaluation*. pp. 246–255.

[83] Mohammad SM, Zhu X & Martin J (2014) Semantic Role Labeling of Emotions in Tweets. In *Proc. of the 5th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*. pp. 32–41.

[84] Demšar J et al. (2013) Orange: Data Mining Toolbox in Python. *J of Machine Learning Research* 14: 2349–2353.

[85] McKinney W (2010) Data Structures for Statistical Computing in Python. In *Proc. of the 9th Python in Science Conf*. pp. 51–56.

Prepoznavanje čustev na Twitterju z uporabo nevronskih mrež

Niko Colnerič

doktorska disertacija
predana
Fakulteti za računalništvo in informatiko
kot del izpolnjevanja pogojev za pridobitev naziva
doktor znanosti
s področja
računalništva in informatike

razširjeni povzetek

Ljubljana, 2019

Uvod Čeprav so splet še pred kratkim sestavljale predvsem statične vsebine, se fokus dandanes usmerja k spletiščem, kjer večino vsebine ustvarjajo uporabniki. Med takšna štejemo vsa družbena omrežja, platforme za mikrobloge, spletne enciklopedije in druge, ki prispevajo k drastičnemu povečanju količine prosto dostopnih vsebin v spletu. Posledično postajajo orodja za razumevanje naravnega jezika ključni del analize teh besedil, saj je njihov obseg pogosto prevelik za ročno analiziranje.

Problem avtomatskega prepoznavanja čustev v naravnem jeziku definiramo takole. Za poljubno vhodno besedilo naj algoritem prepozna, katera čustva je avtor izrazil v njem. Kot je značilno za probleme s področja kategorizacije besedil, je naloga takšnega algoritma, da iz nabora možnih čustev izbere tisto, ki je najbolj izraženo v danem besedilu. Poglejmo nekaj primerov:

- Nekdo mi je včeraj spet ukradel kolo. Upam, da mu poči guma!
- Če samo pomislim na šolo po koncu poletja, me kar zmrazi.
- Sonce, morje, ležalnik in knjiga. Kaj drugega si lahko želiš?
- Že sedaj rabim en prost dan, pa je šele torek.
- Moja služba sploh ni stresna!

Čeprav večina ljudi precej hitro razbere, kakšna čustva so avtorji zgornjih besedil želeli izraziti, je takšen problem težji za avtomatske algoritme, predvsem zaradi pomanjkanja kopice predznanja, ki ga ljudje implicitno upoštevamo pri takšnih odločitvah. Obstajajo pa seveda primeri, ki so nekoliko bolj nedorečeni. Na primer, zadnja poved je lahko resnična trditev ali sarkastična, sámo besedilo pa ne vsebuje dovolj konteksta, da bi med čustvoma lahko ločili.

Uporaba algoritmov za prepoznavanje čustev v besedilih se je izkazala koristna za ocenjevanje javnih mnenj [1], za napovedovanje uspešnosti prodaje knjig [2] in drugih produktov [3]. Raziskovalci opažajo tudi korelacijo med finančno uspešnostjo filmov in čustvi, ki jih spletni uporabniki izražajo, ko govorijo o filmu [4]. Z analizo tvitov napovedujejo prihodke od prodaje vstopnic za kino [5], nekateri raziskovalci pa celo izrabljajo spletne vsebine za napovedovanje cen delnic [6–8].

Dosedanji pristopi k avtomatskemu prepoznavanju čustev besedila najprej pretvorijo v vektorske predstavitve s postopkom vreče besed in tem značilkam večkrat dodajajo

druge, ročno generirane značilke. Tako vektorsko predstavljena besedila so nato vhod v preproste algoritme strojnega učenja, kot sta na primer logistična regresija in metoda podpornih vektorjev, ki nam vrnejo klasifikatorje, sposobne prepoznavanja čustev v besedilih. Zadnje čase takšne preproste pristope prehitevajo pristopi z uporabo nevronskih mrež. Ker te za prepoznavanje čustev v besedilih še niso bile uporabljene, je prvo vodilno vprašanje našega raziskovanja, ali lahko nevronske mreže izboljšajo natančnost prepoznavanja čustev. V ta namen bomo najprej pridelali učne podatke, potem na njih uporabili obstoječe metode, ki jih bomo nato primerjali z rezultati modelov nevronskih mrež.

Poleg tega večina dosedanjih raziskav napoveduje čustva neke določene klasifikacije, čeprav splošnega konsenza glede univerzalnega nabora čustev ni in posledično obstaja več klasifikacij čustev. Naša raziskava se osredotoča na prepoznavanje čustev treh najpopularnejših klasifikacij. Obravnavanje več klasifikacij hkrati najprej podpira njihovo primerjanje, hkrati pa omogoči, da delamo z vsemi hkrati. Zanima nas tudi, ali lahko razvijemo model, ki bo iz skupne vektorske predstavitve besedila lahko prepoznaval čustva različnih kategorizacij.

Klasifikacije čustev Med najpopularnejše kategorizacije čustev, ki se uporabljajo na področju procesiranja naravnega jezika, štejemo Ekmanov nabor osnovnih čustev, Plutchikovo kolo čustev in profil razpoloženjskih stanj (angl. *Profile of Mood States (POMS)*).

Paul Ekman je z opazovanjem obraznih mimik definiral nabor šestih osnovnih čustev [10], za katera trdi, da se pojavljajo pri ljudeh vseh kultur in socio-ekonomskih statusov. To so *jeza*, *gnus*, *strah*, *veselje*, *žalost* in *presenečenje*.

Robert Plutchik je predstavil kolo čustev [11], ki definira osem paroma nasprotnih čustev: *veselje – žalost*, *zaupanje – gnus*, *strah – jeza*, *presenečenje – pričakovanje*. Za vsako izmed njih navede tudi tri različne inačice, ki se razlikujejo po intenziteti. Če je bes ekstremna manifestacija jeze, je neprijetnost njena mila oblika.

POMS [12] je anketni vprašalnik, sestavljen iz 65 pridevnikov, ki opisujejo različna čustvena stanja, s pomočjo katerih oceni anketirančevo razpoloženjsko stanje. Anketiranec za vsakega izmed pridevnikov na petstopenjski lestvici označi, kako pogosto je takšno čustvo občutil v zadnjem tednu. Odgovori za vse pridevnike se nato združijo v skupine, kjer vsak pridevnik prispeva natančno k eni skupini, in tako dobimo končni sedemdimenzionalni opis anketirančevega razpoloženjskega stanja. Na primer, če smo se počutili ekstremno pozabljivi, bo to pozitivno prispevalo k rezultatu dimenzije zmede-

nost. POMS torej vrne oceno za teh sedem dimenzij: *jeza*, *depresija*, *izčrpanost*, *vitalnost*, *napetost*, *zmedenost* in *prijaznost*[1].

Učni podatki Ker pristopamo k prepoznavanju čustev z metodami nadzorovanega strojnega učenja, moramo za svoje eksperimente pridobiti označene učne primere. Naši podatki vsebujejo tvite, ki so bili neprekinjeno zbirani preko Twitterjevega vmesnika med avgustom 2008 in majem 2015. Množica obsega približno 73 milijard tvitov in v nekompresiranih tekstovnih datotekah zasede 17 TB prostora na disku. Tvite smo uvozili v gručo 40 strežnikov, na katerih je nameščen distribuiran datotečni sistem, in razvili aplikacijo za iskanje, ki lahko preišče celotno množico podatkov v uri in pol.

Upoštevajoč velikost naše množice podatkov, je ročno označevanje vsebine vsekakor prezahtevno in zato potrebujemo vsaj delno avtomatski postopek. K sreči ima tak mehanizem Twitter že vgrajen, saj podpira avtorjevo samooznačevanje vsebine v obliki tako imenovanih tematskih oznak (angl. *hashtags*), te pa so med uporabniki zelo priljubljene. Za izgradnjo naše učne množice bomo tako iskali tvite, ki vsebujejo tematske oznake, ki pripadajo čustvenim besedam, nato pa bomo te besede odstranili iz vsebine in jih uporabili kot ciljne spremenljivke. Naši klasifikatorji se bodo tako učili, kako napovedati čustvene oznake iz preostanka vsebine posameznega tvita. Za Ekmanovo in Plutchikovo klasifikacijo iščemo tvite, ki vsebujejo tematsko oznako, pripadajočo šestim oziroma osmim osnovnim čustvom, ki jih kategorizaciji definirata. Za POMS najprej poiščemo vse tvite, ki vsebujejo tematsko oznako, pripadajočo kateremu izmed 65 pridevnikov, tvite pa nato združimo v skupine, tako kot to definira POMS. Naši klasifikatorji bodo tako napovedovali šest kategorij razpoloženjskega stanja in ne neposredno 65 pridevnikov. Tako pridelamo tri podatkovne množice tvitov skupaj z njihovimi oznakami čustev.

Žal pa vsi tako pridobljeni tviti niso primerni za učenje. Na primer, med njimi najdemo tudi primere, kot sta:

- #Strah ne sme biti razlog za nasprotovanje novemu zakoniku!
- #praznovanje #torta #baloni #sveče #presenečenje instagram.com/p/Y1F20

V prvem primeru tematska oznaka *#Strah* dejansko ni uporabljena kot oznaka vsebine, ampak kot ključni del povedi. Drugi primer pa vsebuje samo tematske oznake, ki

[1] V eksperimentih bomo upoštevali samo prvih šest dimenzij, saj se dimenzija *prijaznost* šteje za prešibko za zanesljivo ocenjevanje [13].

označujejo sliko na Instagramu in ne tekstovne vsebine tvita. Da odstranimo takšne tvite iz naše učne množice, uporabimo naslednji hevristični metodi. Najprej izračunamo globino tematske oznake kot razmerje med številom besed, ki se pojavijo pred oznako, in številom vseh besed. Tviti, ki imajo to globino precej nizko, zelo verjetno uporabljajo oznako kot del stavka in so zato odstranjeni. Da odstranimo tvite drugega tipa, definiramo koncept pomenske besede kot besede, ki ni specifična za Twitter (tematske oznake, imena uporabnikov, URL-ji) ter ni številka ali ločilo. Nato izračunamo delež pomenskih besed kot kvocient med številom pomenskih besed in številom vseh besed v tvitu ter odstranimo tvite z nizkim deležem pomenskih besed. Nazadnje odstranimo tudi vse retvite in duplikate. Tako prečiščena množice vsebuje približno 535.000 primerov po Ekmanovi klasifikaciji, 800.000 primerov po Plutchikovi in 6,5 milijona primerov za POMS. Za konec poglejmo še nekaj primerov, ki jih uporabljamo za učenje:

- Najraje bi kričal na ves glas #jeza
- Tudi po štirih mesecih mi misel nate orosi oči #žalost
- Ponovno gledam Sopranove od začetka do konca #veselje

Vsako izmed treh podatkovnih množic razbijemo na tri podmnožice: učno (60 % primerov), validacijsko (20 % primerov) in testno (20 % primerov). Učna in validacijska bosta uporabljeni za nastavljanje parametrov naših modelov. Vsi parametri so izbrani glede na natančnost na validacijski množici, ko je model naučen na učni množici. Ko nastavimo vse parametre modelov, te uporabimo, da naučimo model na kombinaciji učne in validacijske množice. Ta model nato samo enkrat testiramo na testni množici, katere oznake niso bile nikoli razkrite nobenemu učnemu algoritmu.

Klasične metode Najprej bomo uporabili obstoječe metode, da ugotovimo, kakšna je njihova natančnost na naših podatkih. Ker je jezik na Twitterju precej neformalen, uporabimo dva načina normalizacije. Prvi bo vseboval vse besede v vseh oblikah, kot se pojavijo v besedilih. Drugi nekatere besede združi v eno: vse omembe uporabnikov zamenjamo z besedo <*uporabnik*>, vse URL-je z <*url*>, vse številke z <*številka*>, vse tri ali več zaporednih znakov znotraj besede zamenjamo z enim znakom (torej gooool zamenjamo z gol) in vse velike črke zamenjamo z malimi. Za vsak način normalizacije eksperimentiramo z modelom, ki uporablja samo besede, ter modelom, ki poleg besed uporablja tudi pare besed. Poleg modelov vreče besed testiramo tudi modele latentnega

semantičnega indeksiranja. Vsakega izmed zgornjih štirih modelov vreče besed transformiramo v semantični prostor in ohranimo toliko dimenzij, da pojasnimo 70 % variance v podatkih.

Na tako transformiranih besedilih primerjamo štiri algoritme strojnega učenja: naivni Bayes, logistična regresija, naključni gozdovi in metoda podpornih vektorjev. Rezultate algoritma, ki se bo izkazal za najnatančnejšega, bomo uporabili za primerjavo z nevronskimi mrežami.

Nevronske mreže Eksperimentirali bomo z dvema tipoma nevronskih mrež, rekurenčnimi in konvolucijskimi, ter z dvema granularnostma na vhodu: besedami ali znaki. Kadar bodo na vhodu v nevronsko mrežo besede, uporabimo obstoječe vpetje besed GloVe [16], ki je bilo naučeno na podatkih s Twitterja. Tekom treniranja nevronskih mrež to vpetje ali nadalje treniramo ali ne; to smatramo kot dodaten parameter, ki ga izberemo s pomočjo validacijske množice. Kadar bodo na vhodu znaki, učimo njihovo vpetje od začetka, začenši z naključno inicializacijo, saj ne poznamo nobenega prosto dostopnega vpetja znakov. Glavna prednost učenja neposredno na znakih je, da ne potrebuje nikakršnega procesiranja besedil. Vhodno besedilo neposredno znak za znakom podajamo nevronski mreži, ki mora med drugim sama ugotoviti, kako znaki tvorijo besede, da lahko iz njih prepoznava čustva.

Arhitektura rekurenčnih nevronskih mrež je naslednja. Vpetju besed ali znakov sledi rekurenčna plast in za njo po potrebi še ena do dve rekurenčni plasti. Vse rekurenčne plasti skupaj tvorijo zadnji skriti nivo nevronske mreže, ki je preko polno povezane plasti povezan z izhodom nevronske mreže. Pri rekurenčnih plasteh smo testirali dva tipa celic: LSTM [18] in GRU [19].

Arhitektura konvolucijskih mrež za namene klasifikacije besedil, kot je bila predstavljena nedavno [21], je nekoliko preprostejša. Vhodni plasti sledi plast eno-dimenzionalne konvolucije. Za njo poiščemo maksimalne vrednosti skozi čas (angl. *max pooling over time*) in te preko polno povezane plasti povežemo z izhodom nevronske mreže.

Tako v rekurenčnih kot konvolucijskih nevronskih mrežah uporabljamo več plasti za preprečevanje čezmernega prilagajanja učnim podatkom (angl. *dropout layers*). Nevronske mreže treniramo s postopkom gradientnega spusta. Za rekurenčne mreže uporabljamo metodo RMSprop [25], za konvolucijske pa Adam [28]. Za določitev potrebnega števila dob učenja (angl. *epochs*) uporabimo metodo predčasne ustavitve (angl. *early stopping*), ki ustavi učenje, kadar se natančnost na validacijski množici ne poveča v petih

zaporednih dobah.

Prenosljivost rezultatov učenja Tako pri rekurenčnih kot konvolucijskih mrežah lahko zadnji skriti nivo, tik pred izhodom mreže, smatramo kot vpetje besedila v vektor fiksne dolžine. Zanima nas, kako splošne so takšne predstavitve besedil. Natančneje, ali je takšna predstavitev, kot smo jo naučili tekom učenja modela za napovedovanje ene klasifikacije čustev, primerna tudi za napovedovanje čustev druge klasifikacije? Odgovor na to vprašanje nam pojasni, do kakšne mere so predstavitve besedil na zadnjem skritem nivoju splošne in potencialno uporabne pri drugih problemih.

V ta namen zasnujemo naslednji eksperiment. Najprej nevronsko mrežno naučimo na klasifikaciji A. Nato bomo te parametre nevronske mreže uporabili za napovedovanje klasifikacije B. Vzamemo vse parametre, ki vodijo do zadnjega skritega nivoja, in jih iz mreže A prekopiramo v mrežo B ter fiksiramo. Nato mrežo B učimo, vendar pri tem dovolimo spreminjanje samo parametrov zadnjega polno povezanega nivoja. Tako je mreža B prisiljena uporabiti isti način preslikave vhoda v zadnji skriti nivo, kot ga je uporabljala mreža A. Nato natančnost tako trenirane mreže B primerjamo z natančnostjo mreže B, če bi jo trenirali od začetka z naključno inicializacijo in pustili spreminjanje vseh parametrov. Če opazimo, da je uporaba vpetja mreže A drastično škodovala natančnosti mreže B, lahko iz tega sklepamo, da je vpetje mreže A specifično za kategorizacijo A in ni primerno za napovedovanje kategorizacije B. Nasprotno, če se natančnost skoraj ne bi zmanjšala, potem je vpetje A dovolj splošno, da lahko iz njega napovedujemo tudi kategorizacijo B.

Skupno učenje Zadnja skupina eksperimentov se ukvarja z razvojem skupnega modela, torej modela, ki zna iz skupne predstavitve zadnjega skritega nivoja nevronske mreže napovedovati vse tri klasifikacije čustev. Arhitektura takšnega modela je enaka kot arhitekture nevronskih mrež v prejšnjih poglavjih, le da imamo za zadnjim skritim stanjem tri polno povezane nivoje do treh izhodov nevronske mreže – za vsako klasifikacijo imamo svoj izhod. Take arhitekture, ki učijo več nalog hkrati, večkrat vodijo do boljše generalizacije, manj čezmernega prilagajanja učnim podatkom in natančnejših modelov.

Takšni skupni modeli pa potrebujejo poseben način učenja. Ker so naši učni podatki sestavljeni iz treh podatkovnih množic, kar posledično pomeni, da imamo določen učni primer označen samo za eno klasifikacijo in ne za vse tri, takšnega modela ne moremo neposredno učiti s standardnimi pristopi, saj za izhode, za katere ne poznamo anotacij, ne moremo izračunati gradienta. Zato takšne modele učimo drugače, na primer

Algoritem 5 Metoda izmenjujočih se paketov avtorjev Colloberta in Westona [30].

Vhod: M = $\{m_1, m_2, \ldots, m_n\}$ ▷ podatkovne množice
MODEL ▷ model nevronske mreže
ST_EPOHOV ▷ maksimalno število epohov
ST_POSODOBITEV ▷ število posodobitev znotraj enega epoha
Izhod: MODEL ▷ naučen model nevronske mreže

```
1: for i = 1 → ST_EPOHOV do
2:     for j = 1 → ST_POSODOBITEV/|M| do
3:         for m ∈ M do
4:             p ← naslednji_ucni_paket(m)
5:             posodobi_glede_na_paket(p, MODEL)
6:     for m ∈ M do
7:         /* oceni natančnost na učni in validacijski množici */
8:     if kriterij za predčasno ustavitev izpolnjen then
9:         prekini
```

tako, kot sta predstavila Collobert in Weston [30]. Njun pristop iterira čez podatkovne množice in iz njih izbira pakete učnih primerov. Za nek paket potem posodobi parametre nevronske mreže tako, da posodobi vse skupne parametre in vse parametre, ki pripadajo izhodu za trenutno klasifikacijo. Parametre, ki pripadajo izhodom drugih klasifikacij, pusti nedotaknjene. Intuitivno izmenjuje podatkovne množice in tako nauči vse parametre nevronske mreže, zato ta način mi imenujemo metoda *izmenjujočih se paketov*. Psevdokoda je predstavljena v algoritmu 5, kjer uporabljamo naslednji metodi: *naslednji_ucni_paket(m)* za dano podatkovno množico m vrne naslednji učni paket, *posodobi_glede_na_paket(p, MODEL)* pa posodobi parametre modela *MODEL* z enim korakom gradientnega spusta glede na učne primere v paketu p.

Metoda izmenjujočih se paketov tako tekom učenja uporabi enako število učnih primerov iz vsake podatkovne množice. V našem primeru, kjer se učne množice drastično razlikujejo po velikosti, se tak način učenja izkaže za pomanjkljivega predvsem za največjo podatkovno množico. V času, ko model osvoji manjši dve množici, še ne obvlada največje, kar posledično pomeni slabšo natančnost na največji množici. Da odpravimo to pomanjkljivost, predlagamo nov način učenja skupnih modelov. Glavna ideja

našega pristopa je, da mora biti algoritem učenja sposoben dinamično prilagajati, koliko primerov iz posamezne podatkovne množice uporabi glede na napredek učenja. Iz podatkovnih množic, kjer je napredek učenja boljši, mora jemati manj učnih primerov kot iz podatkovnih množic, kjer je napredek slabši. Tako bomo med učenjem posvetili več pozornosti podatkovnim množicam, kjer je napredek slabši. Ideja za ocenjevanje napredka učenja izvira iz tehnike, ki jo raziskovalci večkrat uporabljamo ročno. To je opazovanje razlike med natančnostjo modela na učni in validacijski množici. Kadar model prehaja v fazo čezmernega prilagajanja podatkom, bo točnost na učni množici rasla, medtem ko bo točnost na validacijski nespremenjena oziroma bo začela padati. Posledično, večja ko je razlika, bolj je model čezmerno prilagojen učnim podatkom. Naš algoritem uporablja to intuicijo za usmerjanje vzorčenja učnih paketov. Raje kot da iteriramo čez učne množice, bomo te vzorčili uteženo glede na njihov napredek učenja. Psevdokoda našega pristopa, ki ga imenujemo *uteženo vzorčenje paketov*, je predstavljena v algoritmu 6. Metoda *nakljucno_izberi(M, utezi)* bo iz množice podatkovnih množic $\mathcal{M}$, upoštevajoč *utezi*, uteženo izbrala eno podatkovno množico. Metodi *natancnost_ucni(m)* in *natancnost_validacijski(m)* pa izračunata natančnost modela na učnem oziroma validacijskem delu množice m.

Naš algoritem skozi iteracije spreminja uteži vzorčenja podatkovnih množic. Ker so te uteži lepo konvergirale, smo za učenje modela v testnem načinu za vsako množico uporabili kar njeno povprečje vzorčnih verjetnosti skozi vse dobe učenja.

Rezultati Izmed klasičnih pristopov se je kot najnatančnejši model izkazala logistična regresija na normaliziranih frazah besed. To potrjuje tako koristnost normalizacije kot dodajanja besednih fraz v prostor značilk. Pri nevronskih mrežah se pristopi z znaki na vhodu izkažejo kot najboljši, če le imamo dovolj podatkov. Prednost takih pristopov je predvsem v tem, da ne potrebujejo nobenih jezikovno odvisnih orodij, kot je na primer lematizator, ter so kot taki preprosto prenosljivi na druge jezike. Na podatkovni množici POMS, ki je največja, ti vedno premagajo pristope na podlagi besed, med tem ko se na ostalih, manjših dveh to zgodi v nekaterih primerih. Oba tipa celic pri rekurenčnih mrežah, LSTM in GRU, se izkažeta kot primerljiva. Pri primerjavi rekurenčne arhitekture s konvolucijsko opazimo, da je rekurenčna večinoma nekoliko boljša, vendar razlike niso velike. Primerjava rezultatov klasičnih pristopov in nevronskih mrež razkrije, da nevronske mreže vedno dajejo nekoliko boljše rezultate kot klasični pristopi, vendar razlike žal niso tako velike, kot smo pričakovali. Vendar pa prinašajo tudi druge prednosti, saj

Algoritem 6 Predlagana metoda uteženega vzorčenja paketov.

Vhod: M = $\{m_1, m_2, \ldots, m_n\}$ ▷ podatkovne množice
MODEL ▷ model nevronske mreže
ST_EPOHOV ▷ maksimalno število epohov
ST_POSODOBITEV ▷ število posodobitev znotraj enega epoha
Izhod: MODEL ▷ naučen model nevronske mreže

1: $utezi \leftarrow [1/n, 1/n, \ldots, 1/n]$
2: **for** $i = 1 \rightarrow$ ST_EPOHOV **do**
3: **for** $j = 1 \rightarrow$ ST_POSODOBITEV **do**
4: $im \leftarrow nakljucno\_izberi(\mathrm{M}, utezi)$
5: $p \leftarrow naslednji\_ucni\_paket(im)$
6: $posodobi\_glede\_na\_paket(p, \mathrm{MODEL})$
7: **for** $m \in \mathrm{M}$ **do**
8: /* oceni natančnost na učni in validacijski množici */
9: $napredek \leftarrow natancnost\_ucni(m) - natancnost\_validacijski(m)$
10: $utezi[m] \leftarrow 1/napredek$
11: $utezi \leftarrow utezi/vsota(utezi)$
12: **if** kriterij za predčasno ustavitev izpolnjen **then**
13: $prekini$

nevronske mreže z znaki na vhodu ne zahtevajo razčlenjevanja besedila, normalizacije besed in generiranja značilk. Spomnimo, da za najboljše rezultate klasičnih pristopov potrebujemo normalizacijo besed in vključevanje besednih fraz. Upoštevajoč nekoliko višjo natančnost in predvsem manjšo vključenost človeka, zaključimo, da so nevronske mreže primernejše za prepoznavanje čustev.

Prenosljivost predstavitev zadnjega skritega stanja nevronskih mrež, kadar so te učene na eni podatkovni množici, se izkaže za neobetavno. Edini primer, kadar je natančnost mreže, ki uporablja isti način preslikave vhoda v zadnji skriti nivo, kot je bil naučen na drugi mreži, primerljiva z natančnostjo učenja celotne mreže, je, kadar prenesemo uteži iz modela za napovedovanje Plutchikovih kategorij na model, ki napoveduje Ekmanove. Vendar ta eksperiment ni smiseln, saj so Ekmanove kategorije podmnožica Plutchikovih. Rezultati prenosljivosti ostalih eksperimentov, torej kadar prvo mrežo učimo na klasifi-

kaciji POMS in jo prenesemo na Ekmanovo ali Plutchikovo klasifikacijo ter prav tako eksperimentov v drugo smer, pokažejo, da so predstavitve zadnjih skritih stanj modelov učenih na eni podatkovni množici precej prilagojene trenutni klasifikaciji, saj opažamo velike izgube pri natančnosti.

Kadar skupni model učimo z znano metodo izmenjujočih se paketov, opazimo, da je natančnost modela pri Ekmanovi in Plutchikovi klasifikaciji primerljiva z natančnostjo posameznih modelov. Natančnost na klasifikaciji POMS pa je približno 10 % slabša od natančnosti posameznega modela. To potrjuje sum, da obstoječa metoda ne upošteva različnih velikosti oziroma kompleksnosti podatkovnih množic, kar vodi v slabšo natančnost večjih oziroma težjih podatkovnih množic. Z uporabo novo predstavljene metode uteženega vzorčenja paketov se natančnost na klasifikaciji POMS dvigne in postane primerljiva z natančnostjo posameznega modela, pri tem pa se natančnost na ostalih dveh množicah ne poslabša. Opazimo, da je verjetnost vzorčenja za POMS večja od vsote verjetnosti ostalih dveh množic. Algoritem je tako več kot polovico učnih primerov črpal iz množice POMS. Primerjava harmoničnih sredin natančnosti čez vse podatkovne množice razkrije, da nova metoda uteženega vzorčenja paketov v veliki večini primerov premaga obstoječo metodo izmenjujočih se paketov. Tudi najnatančnejši skupni model je bil naučen z novo metodo učenja. To potrdi tako smiselnost uteženega vzorčenja kot metodo ocenjevanja napredka učenja. Z novo metodo tako uspemo naučiti model, katerega natančnost je uravnotežena čez vse tri podatkovne množice in je primerljiva z natančnostjo posameznih modelov. Ker skupni model to doseže s prepoznavanjem iz skupne predstavitve zadnjega skritega stanja, verjamemo, da je ta bolj splošna kot skrita stanja posameznih modelov. Splošnost predstavitev zadnjega skritega stanja skupnega modela smo testirali tudi na drugih podatkovnih množicah za prepoznavanje čustev in sentimenta. Opazimo, da smo z uporabo vpetja skupnega modela, ki smo ga nato fiksirali ali dovolili, da se nadalje spreminja tekom učenja, vedno uspeli preseči natančnost nevronske mreže, ki ne uporablja našega vpetja. Preslikava vhodnega besedila v zadnji skriti nivo skupnega modela tako predstavlja univerzalno vpetje, primerno za napovedovanje več klasifikacij čustev hkrati. Ima pa tudi potencial, da izboljša rezultate podobnih napovednih nalog, kot je na primer napovedovanje sentimenta.

Zaključek Ta disertacija obravnava problem avtomatskega prepoznavanja čustev na Twitterju. V ta namen pridelamo verjetno največjo množico podatkovnih primerov, ki izvirajo iz obdobja sedmih let. Naši učni primeri niso omejeni samo na neko tematsko

skupino, na primer finance, ampak raje preizkusimo algoritme v splošnem.

Eksperimentalno pokažemo, da lahko z nevronskimi mrežami izboljšamo natančnost algoritmov prepoznavanja čustev, hkrati pa je potrebna manjša vključenost človeka kot pri klasičnih pristopih. Z uporabo nove metode učenja uspemo naučiti skupni model, sposoben napovedovanja vseh treh klasifikacij iz skupne predstavitve, z natančnostjo, ki je primerljiva z natančnostjo najboljših posameznih modelov.

Ideje za nadaljnje delo vključujejo prepoznavanje in odstranjevanje sarkastičnih tvitov iz podatkovne množice, saj ti zavajajo algoritme. Dodajanje nevtralnih primerov v podatkovno množico bi algoritme prepoznavanja učilo tudi prepoznavanja nevtralnih tvitov, ki predstavljajo zajeten del resničnih podatkov. Čeprav se algoritmi izkažejo kot natančni na avtomatsko označenih podatkih, bi bilo zanimivo njihove napovedi preveriti na novih podatkovnih množicah.

Prispevka k znanosti

- *Nov postopek učenja skupnih nevronskih mrež*

  Predstavimo nov postopek učenja nevronskih mrež, ki napovedujejo več nalog hkrati. Postopek je posebej primeren, kadar se podatkovne množice razlikujejo v velikosti ali kompleksnosti. Motivacija prihaja iz načina, kako raziskovalci ocenjujejo stopnjo čezmernega prileganja učnim podatkom. Razliko med natančnostjo na učni in validacijski množici smatramo kot oceno napredka učenja ter jo uporabimo kot vodilo za usmerjanje postopka vzorčenja učnih primerov. Pokažemo, da s takšnim postopkom učenja vidno izboljšamo natančnost skupnega modela, predvsem na naši največji podatkovni množici (POMS), ki je z uporabo obstoječih metod kazala slabše rezultate.

- *Univerzalno vpetje besedil, primerno za napovedovanje čustev*

  Z razvojem skupnega modela, ki prepoznava tri kategorizacije iz skupne predstavitve, pridobimo univerzalno vpetje besedil, ki je primerno za napovedovanje več kategorizacij čustev.

Tehnični prispevek

- *Javno dostopni modeli za prepoznavanje čustev*

  Najnatančnejše izmed naših modelov smo naredili prosto dostopne v programskem jeziku Python in programu Orange (glej poglavje 6), kar poenostavi primerjavo prihodnjih raziskav z našimi modeli.