Emotion Recognition on Twitter: Comparative Study and Training a Unison Model

Niko Colnerič and Janez Demšar

Abstract—Despite recent successes of deep learning in many fields of natural language processing, previous studies of emotion recognition on Twitter mainly focused on the use of lexicons and simple classifiers on bag-of-words models. The central question of our study is whether we can improve their performance using deep learning. To this end, we exploit hashtags to create three large emotion-labeled data sets corresponding to different classifications of emotions. We then compare the performance of several word- and character-based recurrent and convolutional neural networks with the performance on bag-of-words and latent semantic indexing models. We also investigate the transferability of the final hidden state representations between different classifications of emotions, and whether it is possible to build a unison model for predicting all of them using a shared representation. We show that recurrent neural networks, especially character-based ones, can improve over bag-of-words and latent semantic indexing models. Although the transfer capabilities of these models are poor, the newly proposed training heuristic produces a unison model with performance comparable to that of the three single models.

Index Terms—Emotion Recognition, Text Mining, Twitter, Recurrent Neural Networks, Convolutional Neural Networks.

1 Introduction

The amount of user-generated content on the web grows ever more rapidly, mainly due to the emergence of social networks, blogs, micro-blogging sites and a myriad of other platforms that enable users to share their personal content. Unlike objective and factual professional publishing, user-generated content is richer in opinions, feelings and emotions. These online expressions can have various practical applications. They have been used to predicted stock market fluctuations [1], book sales [2], or movie's financial success [3]. Due to the vast number of texts, manual inspection for emotion classification is infeasible, hence the need for accurate automatic systems. Although in many cases people can easily spot whether the author of a text was angry or happy, the task is quite challenging for a computer — mainly due to the lack of background knowledge that is implicitly considered by humans.

Given some text, emotion recognition algorithms detect which emotions the writer wanted to express when composing it. To treat this problem as a special case of text classification, we need to define a set of basic emotions. Although emotions have long been studied by psychologists, there is no single, standard set of basic emotions. Therefore, we decided to work with three classifications that are the most popular, and have also been used before by the researchers from computational linguistics and natural language processing (NLP). Paul Ekman defined six basic emotions by studying facial expressions [4]. Robert Plutchik extended Ekman's categorization with two additional emotions and presented his categorization in a wheel of emotions [5]. Finally, Profile of Mood States (POMS) is a psychological instrument that defines a six-dimensional mood state representation [6]. Each dimension is defined by a set of emotional adjectives, like *bitter*, and the individual's mood is assessed by how strongly (s)he experienced such a feeling in the last month.

Majority of previous studies predict either Ekman's or Plutchiks's classifications, while POMS's adjectives had only been used in simple keyword spotting algorithms [1]. We are not aware of any studies that tackle the problem of *predicting* POMS's categories from the text. Methodologically, they mainly used simple classification algorithms, like logistic regression or support vector machines, on top of word and n-gram counts, and other custom engineered features (capturing the use of punctuation, the presence or absence of negation, and counts of words from various emotion lexicons) [7], [8], [9], [10].

Deep learning has recently shown a lot of success in the field of NLP: it has been used for improving sentiment analysis [11], opinion mining [12] and other tasks like part-of-speech tagging, chunking, named entity recognition, and semantic role labelling [13]. Radford et. al [14] even reported on discovering the sentiment unit while training to generate reviews. Despite these successes, deep learning approaches for emotion recognition are almost non-existent.

Previous work generally studied only one emotion classification. Working with multiple classifications simultaneously not only enables performance comparisons between different emotion categorizations on the same type of data, but also allows us to develop a single model for predicting multiple classifications at the same time.

We studied the following questions.

- Can deep learning methods improve the performance of emotion detection?
- How transferable are deep learning's internal representations? For example, is a sentence embedding

• *N. Colnerič and J. Demšar are with the Faculty of Computer and Information Science, University of Ljubljana, Slovenia. E-mail: {niko.colneric, janez.demsar}@fri.uni-lj.si*

The work was partially done while visiting Stanford University.
The study was supported by Slovenian Research Agency; program P2-0209.

learned on Ekman's categories sufficient for predicting POMS's categories?
- Can we train a unified neural network for predicting all three emotion classifications at the same time?
- Are POMS's categories harder to predict than Ekman's or Plutchik's?

The next section describes three psychological classifications of emotions. The third section outlines the data collection and filtering. To obtain an emotion-labeled training data set, we exploited the presence of hashtags, a self-annotation mechanism frequently used on Twitter. In particular, we used the hashtags that, according to the three classifications, directly correspond to particular emotions. We further filtered the tweets with heuristics aimed at increasing the chances that the hashtag found in the tweet actually reflects the author's emotions.

The fourth section focuses on algorithms. We start with commonly used classifiers on bag-of-words and latent semantic indexing models that serve as the baseline that we try to surpass with deep learning. We use recurrent and convolutional neural network models, into which we feed either words or individual characters. We describe how to transfer models trained with one classification of emotions to another classification. Finally, we show a novel algorithm for fitting a common network for the three classifications with different complexity and very different data set sizes. The fifth section shows empirical results and debate on ideas for future work. Finally, we discuss the related work and highlight the outcomes in concluding section.

2 Emotion Classifications

Paul Ekman [4] studied facial expressions to define a set of six universally recognizable basic emotions: *anger*, *disgust*, *fear*, *joy*, *sadness* and *surprise*.

Robert Plutchik [5] defined a wheel-like diagram with a set of eight basic, pairwise contrasting emotions; *joy – sadness*, *trust – disgust*, *fear – anger* and *surprise – anticipation*. We consider each of these emotions as a separate category, and we disregard different levels of intensities that Plutchik defines in his wheel of emotions.

Profile of Mood States [6] is a psychological instrument for assessing the individual's mood state. It defines 65 adjectives that are rated by the subject on the five-point scale. Each adjective contributes to one of the six categories. For example, feeling *annoyed* will positively contribute to the *anger* category. The higher the score for the adjective, the more it contributes to the overall score for its category, except for *relaxed* and *efficient* whose contributions to their respective categories are negative. POMS combines these ratings into a six-dimensional mood state representation consisting of categories: *anger*, *depression*, *fatigue*, *vigour*, *tension* and *confusion*. Since POMS is not publicly available, we used the structure from Norcross *et al.* [6], which is known to closely match POMS's categories. We supplemented it with additional information from the BrianMac Sports Coach website[1]. Comparing to the original structure, we discarded the adjective *blue*, since it only rarely corresponds to an emotion and not a color, and word-sense disambiguation tools were unsuccessful at distinguishing between the two meanings. We also removed adjectives *relaxed* and *efficient*, which have negative contributions, since the tweets containing them would represent counter-examples for their corresponding category.

For each category we used the following adjectives:

- **anger**: angry, peeved, grouchy, spiteful, annoyed, resentful, bitter, ready to fight, deceived, furious, bad tempered, rebellious,
- **depression**: sorry for things done, unworthy, guilty, worthless, desperate, hopeless, helpless, lonely, terrified, discouraged, miserable, gloomy, sad, 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,
- **confusion**: forgetful, unable to concentrate, muddled, confused, bewildered, uncertain about things.

From now on, we will refer to these classifications as Ekman, Plutchik and POMS.

3 Data

We first describe the procedure used for annotating our collection of tweets and then present some basic statistics of the resulting data sets.

3.1 Labelling Examples by Distant Supervision

Since we wish to exploit our data trove of 73 billion tweets, any approach requiring manual annotation is infeasible. We instead assume that hashtags in most cases reveal the author's true emotion, with an additional advantage of being selected by the author her/himself. The idea of exploiting hashtags as annotations has already been proven successful in many recent studies, such as for sentiment classification [15], [16], [17], detecting sarcasm [18], [19], classifying emotions [7] and even for studying personality traits [10].

We search among English tweets for exact matches of six emotional hashtags for Ekman (*#anger*, *#disgus*, *#fear*, *#joy*, *#sadness* and *#surprise*); eight for Plutchik (*#joy*, *#sadness*, *#trust*, *#disgust*, *#fear*, *#anger*, *#surprise* and *#anticipation*) and 55 adjectives for POMS[2]. Multiword expressions were concatenated to a single-word hashtag, like *#unabletoconcentrate*. We kept only the tweets that contained one or more of these hashtags.

For Ekman's and Plutchik's data sets, the hashtag itself defines the target category. For POMS's adjectives, we set the target category as the mood to which the adjective belongs, as defined in Section 2. For example, all tweets containing either *#angry*, *#grouchy*, *#furious*, etc. are used as training examples for the *anger* category. We will therefore only predict POMS's six mood categories and not 55 adjectives. If classifiers perform well on such data, this also

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

2. Note that original set of 65 adjectives also contains adjectives contributing to the seventh category *friendliness*. Removing these adjectives — since they are considered too weak for valid scoring [6] — along with the one mentioned above yielded a final set of 55 adjectives.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TAFFC.2018.2807817, IEEE Transactions on Affective Computing

TABLE 1
Examples of tweets along with the correct target values our classifiers are trained to detect. 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	×	✓	✓	×	✓	×

TABLE 2
Examples 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 detecting emotions. Such tweets were removed from the training set.

#Fear is not a good reason to oppose #AB230. Education should involve educating humans about being human. That includes sex.
#birthday #sweet #seventeen #cake #tart #ballon #candle #night #asia #surprise #love #happy thnks... instagram.com/p/Y1F20ZALlr/

TABLE 3
Examples of tweets passing all filtering requirements. Note that the models received the tweets' contents without the emotional hashtags.

Why am i suddenly feeling the urge to Scream at the top of my lungs?? #anger #monday #apushsucks
4 months gone and thinking of u still brings tears to my eyes #sadness
Watching the sopranos again from start to finish! #joy

indirectly confirms the sensibility of the factor structure defined by POMS. All emotional hashtags mentioned above — which serve as target variables — were removed from tweets before training to prevent the classifiers from merely recognizing the key hashtags instead of deducting them from the content.

A few tweets contain multiple categories, so experiments include *multiclass* as well as *multilabel* setting. In the former, we will build a single non-binary classifier for predicting the first emotional category, disregarding any other emotional hashtag present later in the tweet. Classifiers in this setting only have to pick the most probable emotion from a set of all possible emotions. In the second setting, we build multiple binary classifiers — one per emotional category. For each of them independently, the classifiers have to provide a decision whether the tweet expresses that emotion or not. Table 1 shows some examples. Multilabel setting accounts for all emotional hashtags that the user provided and is closer to a real-life scenario in which tweets can be non-emotional: the multilabel setting allows a scenario in which all classifiers return a negative prediction.

This resulted in a collection of tweets as well as their emotional categories. However, some of them are not suitable for learning. Some hashtags do not play the role of a label but are rather a part of the tweet's content; see the first example in Table 2. Others contain just hashtags, links, @mentions, numbers and other non-content bearing words, like the second example in Table 2. Since our goal is to develop a model capable of detecting emotions from the *textual content* — and not simply find correlations between certain @mentions or links and emotions, or recommend one hashtag from the presence of the other[3] — such tweets are undesirable in the training data.

We thus apply the following filters. First, we define a *content word* as a token *not* tagged as Twitter/online-specific (e.g. hashtag, @mention, URL) or Miscellaneous (e.g. number, punctuation) according to the Tweet POS tagger [20].

3. Hashtag recommendation belongs to the field of recommendation systems, not natural language understanding.

Using this definition, we calculate the content fraction c as

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

After inspecting the data, we set the threshold to 0.5; i.e. all tweets with $c < 0.5$ were removed from the training data set[4].

To remove tweets containing emotional hashtag as a part of the sentence, we calculated the depth d of the hashtag as

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

Again, after inspecting the data[5], we heuristically set the threshold to 0.9; i.e. all tweets with $d < 0.9$ are filtered out since the hashtags might be a crucial part of the sentence and not a label. Gonzales-Ibanez *et al.* [18] applied a similar procedure, but required that the hashtag appeared at the very end of the tweet, which seems too strict.

Finally, we removed re-tweets and duplicates. Some examples of tweets satisfying all above requirements are presented in Table 3.

3.2 Data Set Statistics

We started with a data set of 73 billion tweets collected between August 2008 until May 2015. The data occupies approximately 17 TB of space in uncompressed form, stored in a Hadoop distributed file system. We have developed a custom map-reduce search application, which takes about an hour and a half to linearly search through the data on a 40 node cluster.

We searched this data set for tweets containing emotional hashtags and satisfying filtering requirements as defined in the previous subsection. We split the resulting data into train (60 %), validation (20 %) and test (20 %) sets[6].

4. We set the threshold by observing a few dozen tweets, which is negligible in comparison with the total number of tweets, so we have not used a separate data set for this.
5. See footnote 4.
6. We cannot publish the data due to legal reasons.

TABLE 4
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

TABLE 5
The number of features of BoW and LSI models for combined train and validation sets using different token normalizations. The name *bigrams* stands for a model consisting of combination of unigrams and bigrams.

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

Train and validation sets served for selecting parameters. Once set, the final models were trained on combined train and validation set, and their performance was assessed on the test set.

To enable a fair evaluation of transfer capabilities, splitting of Ekman's and Plutchik's data sets must be done in parallel since the former is a subset of the latter. That is, Ekman's train, validation and tests sets are subsets of the corresponding Plutchik's data sets. POMS was split independently since its adjectives do not overlap with Ekman's and Plutchik's. The number of tweets in each set is shown in Table 4, and the class distributions are shown in Fig. 1.

Class distributions are quite imbalanced. One reason for this, apart from the peoples' tendencies to express particular kinds of emotions on Twitter, may be the restriction to exact matches of hashtags. Tweets may often contain variations and synonyms for hashtags like *#disgust* and *#vigour*. We leave dealing with synonyms or morphological variations as an interesting idea 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 [7] of emotion recognition on Twitter data.

4 METHODS

We first describe two baseline methods, the common bag-of-words and latent semantic indexing models. This is followed by the description of the neural network models and their adaptation for transfer and unison learning.

4.1 Bag-of-Words & Latent Semantic Indexing Models

To set the baseline performance, we first experimented with common approaches to emotion detection. Within the realm of pure machine learning (as opposed to using, say emotion lexicons), one of the most frequently used approaches is to use simple classifiers on the bag-of-words (BoW) models.

We studied two approaches for transforming raw text into BoW model. *Vanilla BoW* is a model without any normalization of tokens. *Normalized BoW* reduces the dimensionality of feature space by these transformations:

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

The aim of these normalization techniques is to remove the features that are too specific. For each of these two models, we run experiments on counts of unigrams as well as unigrams and bigrams. Hereafter, we will refer to the combination of unigrams and bigrams simply as bigrams.

Tokenization was done using Tweet POS tagger [20]. For each model, we filtered out tokens and bigrams occurring in less than five tweets. These four BoW models served as a basis for experiments with latent semantic indexing (LSI). We determined the number of dimensions to keep so that 70% of the variance was retained. While the threshold comes from [21], the number of retained dimensions is in the range that empirical studies show as appropriate [22]. LSI experiments were only performed for Ekman and Plutchik, since calculating the decomposition for POMS was not possible with the computation resources we had at our disposal. The dimensionality of BoW and LSI models is shown in Table 5.

We experimented with the following classifiers: Support Vector Machines with linear kernel (SVM), Naïve Bayes (NB), Logistic Regression (LogReg) and Random Forests (RF). Regularization parameters for SVM, LogReg, and the number of trees for RF were selected using linear search. Since RF was extremely slow, we built forests with only up to 200 trees. For example, training 200 trees on POMS's data set using bigrams vanilla model took about three days on a computer with 40 cores. All BoW experiments were performed in Python using scikit-learn [23] library and all other parameters[7] were left at their default values (for detailed description refer to the documentation at http://scikit-learn.org/stable/documentation.html).

4.2 Neural Network Models

Among the most popular neural network (NN) architectures, we decided to use recurrent (RNN) and convolutional (CNN) networks. The former were selected since they can naturally handle texts of variable lengths, and latter since they have already shown to be suitable for text classification [24]. We leave the testing of other neural network architectures, like feed-forward ones, for future work.

We experiment with two levels of granularity. In the first approach, we tokenize the tweet's content and then feed a sequence of tokens into the NN. Here the task of the NN is to learn how to combine words to obtain a tweet representation suitable for predicting emotions. Our second setting is

7. *Loss*, *penalty*, *tol* and *max_iter* for SVM; *alpha* for NB; *max_iter*, *solver* and *tol* for LogReg; *criterion*, *max_features*, *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.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TAFFC.2018.2807817, IEEE Transactions on Affective Computing

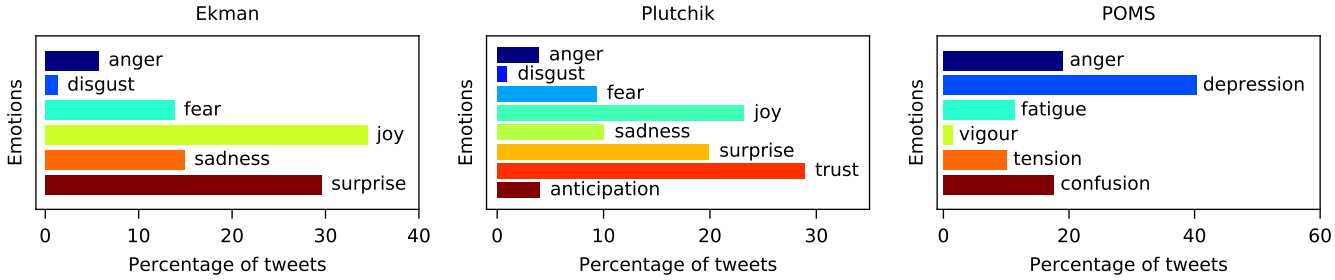

Fig. 1. Class distribution of *train* sets for each emotion classification.

an end-to-end learning approach: instead of preprocessing tweets into tokens, we treat the whole tweet as a sequence of characters and pass characters one by one into the NN. The task of the NN is hence to combine characters into a suitable representation and predict emotions. Note that the NN itself has to learn which sequences of characters form words since space is not treated any differently from any other character.

One benefit of the character-based approach is that it requires no preprocessing and normalization[8]. When working with words we first need a tokenizer to split the tweet into tokens. Next, we have to tackle the problem of normalization. Which morphological variations of words are similar enough that the same token could be used for their representation? For example, should we treat *shop* and *shopping* as the same token? In the character setting all those decisions are left for the NN to figure out.

Sequences of either words or characters are first mapped to vectors, commonly referred to as embedding. For words, we used pre-trained GloVe embedding [25] since it is, to our knowledge, the only embedding fit on Twitter data. Consequently, we also followed the same normalization of tokens as GloVe. Since we have enough data and our task slightly differs from the unsupervised task used for training GloVe embedding, we speculated that further fine-tunning during the training process might improve the embedding. Whether to do this or not was an additional parameter we optimized. For learning from sequences of characters, we used only characters that occurred in our data at least 25 times, which yielded a set of 410 characters including some emoticons and other symbols. For each of them, we trained a character embedding starting from a random initialization.

Embeddings are passed through the dropout layer, followed by either RNN or CNN layers. For RNNs we experimented with three types of layers: the fully recurrent network layers (SimpleRNN), long short term memory (LSTM) and gated recurrent units (GRU), with dropout layers in between. For CNNs we used the architecture from Kim [24]; i.e. one convolutional layer followed by max pooling over time.

The final hidden state representation is passed through one last dropout layer. The final layer is softmax for the multiclass setting and sigmoid for the multilabel setting. The overall architecture is shown in Fig. 2.

8. We only removed URL links due to an anomaly of our crawler. On Twitter, we mainly see shortened URLs but in our data, they had already been expanded. Since this makes tweets much longer than 140 characters we decided to remove all URLs.

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

- GloVe embedding dimensions: 25, 50, 100, 200;
- fine-tune GloVe embedding: no, yes;
- character embedding dimensions: 10, 15, 25, 35;
- dropout for word or character embedding: 0.0, 0.2;
- dropout for softmax: 0.0, 0.5.
- RNN only; layer kind: SimpleRNN, LSTM, GRU;
- RNN only; hidden layers: 1, 2, 3;
- RNN only; neurons per hidden layer: 200, 500, 1000, 2000;
- RNN only; bidirectional layers: no, yes;
- RNN only; dropout between multiple layers: 0.0, 0.2;
- CNN only; feature maps: 100 – 6000 (with step 100)
- CNN only; feature maps activation: relu, tanh
- CNN only; kernel size: 1 – 20

The dropout rates for softmax come from Srivastava *et al.* [26] and for embedding from Zaremba *et al.* [27]. We used RMSProp optimizer for RNNs, Adam for CNNs, a batch size of 128 and early stopping with the patience of 5. All NN experiments were conducted in Python using Keras [28] library.

4.3 Transfer Learning

After selecting the best models and their parameters, we test their transfer capabilities and generality. We investigated whether the final hidden state representation — which can be considered as a projection of the tweet's content into a lower dimensional space — is suitable only for the task for which it was trained or is it sufficient also for predicting other emotion categorizations. We take a model up to the final hidden layer and then re-train the final softmax or sigmoid layer on another data set. In this way, we re-use the embedding from one data set for making predictions on the other. Note that since we are copying weights of one model to the other, we are also forced to use a common model architecture; i.e. the number of neurons, layers, type of layers, number of feature maps, kernel size, etc.

The intuition behind these experiments is that if the final hidden state representation can be considered as a general lower dimensional representation suitable for predicting emotions, then the one trained on Ekman might also suffice for predicting POMS's categories. However, if the performance of such trained model is drastically worse than that of a model initially trained on POMS, this would indicate that final hidden states representations are specifically tunned for particular categorization of emotions.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TAFFC.2018.2807817, IEEE Transactions on Affective Computing

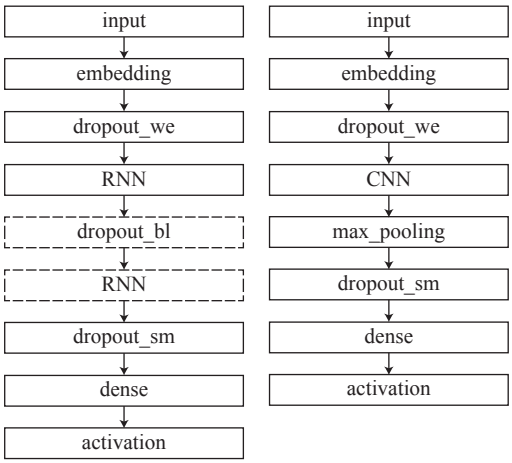

Fig. 2. The architecture of our RNN (left) and CNN (right) models. Dashed boxes are only present in multi-layer architectures. The RNN figure corresponds to a two-layer architecture, while the three-layer architecture includes another pair of *dropout_bl* and *RNN* layers.

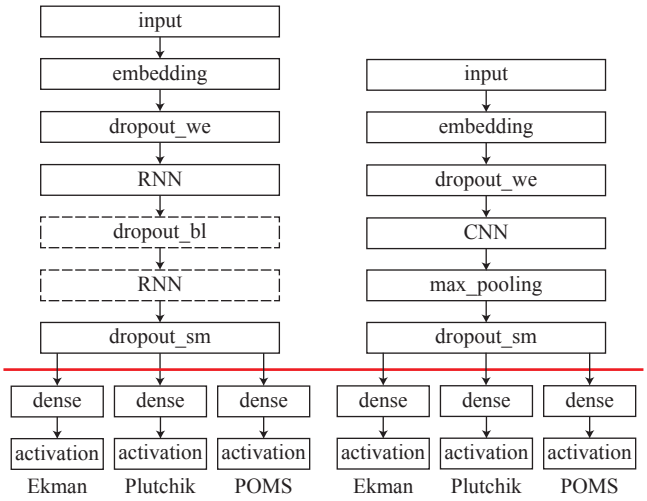

Fig. 3. The architecture of RNN (left) and CNN (right) unison models. Everything above the red line is shared for three emotion classifications; i.e. models use the same word or character embedding as well as the whole RNN or CNN layer. On top of common representation we use one softmax (or sigmoid) layer for each emotion classification.

4.4 Unison Learning

The final set of experiments tests whether it is possible to develop a common model. We define the *unison model* as a model able of predicting all three emotion classifications while sharing all the parameters that project the input tweet into a final hidden state representation. The utility of such model is at least threefold. First, sharing parameters will hopefully lead to a model whose final hidden state representations are more general. The existence of such hidden state — that could be used to predict various emotional classification — is an indication that there exists a general emotion representation, which could be the starting point for investigating the interdependence between emotional classifications. Second, as is believed for multitask learning approaches, introducing these additional signals during the training of a model could lead to better performance [29]. Finally, when applying such model, we get predictions for all classifications in approximately the same computation time a single model would require for one classification.

To build the unison model we propose the following architecture. We have common embedding, followed by a common NN layer. After the final hidden state representation of the NN, there are three different softmax (for multiclass setting) or sigmoid (for multilabel setting) layers, each predicting one of the three classifications. This architecture, presented in Fig. 3, is learning a low-dimensional embedding that is informative enough for predicting all three categorizations at once.

A similar idea was presented by Collobert and Weston [30]; however, their tasks were not closely related and consequently they only shared a word embedding. Contrary, our tasks are related and hence it makes sense to try sharing the whole RNN or CNN layer as well.

In general, similar multitask learning models are trained with back-propagation by summing the gradients corresponding to each of the tasks. This requires that each training example is labeled for all classifications in the unison model. However, we have three data sets that only partially overlap. Consequently, for each training instance, we possess its classification according to only one of the three emotion classifications[9].

Collobert and Weston [30] presented an approach to training such models. The idea is to iterate over all training tasks, each time picking a random example. The neural network weights are then updated according to gradients with respect to the current example. Note that they only update the shared part of the neural network and the part corresponding to the current task. Parts of the model that are not shared and correspond to other tasks are left untouched. Instead of working with only one example at a time, we use batches of multiple examples. We also add the use of early stopping into the training iteration. Due to working with multiple data sets at the same time, early stopping will monitor the average validation accuracy across all data sets. We present this training strategy, referred to as *alternate batches (AB)* strategy, in Algorithm 1.

Note that all data sets on the input have already been split into train and validation part and that function *next_train_batch(x)* returns the next batch from the training part of the data set x — it can be thought of as an endless loop over training batches of x. Function *train_on_batch(b, m)* accepts training batch b and model m and performs one update of model m according to data in b.

Initial experiments with alternate batches strategy showed poor results. Sampling examples from all data sets with the same proportions ignores the different sizes of data sets and the fact that some data sets might be harder to model than others.

We propose an alternative training strategy. To train the model for all data sets in parallel, we take more training examples from the data set on which the progress is slower. The progress estimation is based on the technique that is commonly employed by researchers — i.e. observing the accuracy of the model through training iterations. In each

9. With exception of tweets for Ekman for which we also know the Plutchik's category.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TAFFC.2018.2807817, IEEE Transactions on Affective Computing

Algorithm 1 Alternate Batches strategy by Collobert and Weston [30].

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

for $epoch = 1 \rightarrow EPOCHS$ **do**
  for $update = 1 \rightarrow UPDATES/|DS|$ **do**
    for $ds \in DS$ **do**
      $b \leftarrow next\_train\_batch(ds)$
      $train\_on\_batch(b, MODEL)$
  for $ds \in DS$ **do**
    /* evaluate model on train and validation set */
  if early stopping criteria met **then**
    $break$

iteration, we evaluate the model's accuracy on train and validation set and calculate their difference. When overfitting, the accuracy on the training set still grows while the accuracy on the validation set stagnates. Hence, we believe that this difference, which is small at first and grows bigger the closer we are to the point of overfitting, is indicative of the training progress. We treat the difference between accuracies on train and validation sets as our progress estimate and use it when sampling next training instances: instead of sampling instances uniformly from each data set, we sample with weights based on the progress estimates. By sampling more training instances from data sets whose progress is slower, we boost the fitting to those data sets[10]. The sampling probabilities are inversely proportional to the progress estimate; they are re-calculated after each evaluation and change throughout the training process according to the current progress estimates. The whole training heuristic, which we refer to as *weighted sampling batches (WSB)* strategy, is presented in Algorithm 2. To test the utility of our progress estimates, we also compare WSB with the strategy that samples training batches according to data set sizes (*WSBDS*) instead of progress estimates.

Note that function *random_choice(DS, weights)* samples a data set from *DS* in a weighted manner according to sampling weights given as *weights*. Functions *train_acc(ds)* and *val_acc(ds)* return the accuracy of the model on the training and validation part of the data set *ds* correspondingly.

Finally, note that the weighted sampling batches strategy, as presented in Algorithm 2 requires a labelled validation set. The algorithm described above is run on training and validation data to establish the sampling probabilities. In the actual test run on the hitherto unseen test data for which the class labels are not revealed to the algorithm, the weights are fixed at the average values computed when training on the validation data. We can thus consider setting the sampling weights to be a part of fitting the parameters of the learning algorithm. In the test run, we do not evaluate the model after each epoch but only after the training process has stopped.

We decided to use fixed sampling probabilities through all iterations of the test run since our sampling probabilities

10. The procedure is remotely similar to boosting, except that it works on the whole data sets instead of on individual instances.

Algorithm 2 Proposed Weighted Sampling Batches strategy.

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

$weights \leftarrow [1/n, 1/n, ...1/n]$
for $epoch = 1 \rightarrow EPOCHS$ **do**
  for $update = 1 \rightarrow UPDATES$ **do**
    $ds \leftarrow random\_choice(DS, weights)$
    $b \leftarrow next\_train\_batch(ds)$
    $train\_on\_batch(b, MODEL)$
  for $ds \in DS$ **do**
    /* evaluate model on train and validation set */
    $progress \leftarrow train\_acc(ds) - val\_acc(ds)$
    $weights[ds] \leftarrow 1/progress$
  $weights \leftarrow weights/sum(weights)$
  if early stopping criteria met **then**
    $break$

nicely converged, as we will see in Section 5.4. If sampling probabilities plots in the train-validation run were not so flat, then in each iteration of the test run we could sample according to the sampling probabilities corresponding to the same iteration of train-validation run.

5 Results & Discussion

Experimental results are organized into four sections. In Section 5.1 we observe the baseline performance of BoW and LSI models. Section 5.2 compares these with the performance of RNNs and CNNs. Results of transfer and unison models are given in Sections 5.3 and 5.4, respectively.

Before discussing the results we emphasize that all models' parameters, including the sampling probabilities in the case of weighted sampling batches strategy, were selected according to the performance on the validation set when the model was trained on the train set. Once all parameters were set, we used them when training models on the combined train and validation set. These models were then evaluated only once on the test set consisting of examples that have not been seen during the training or parameter selection process. All subsequent subsections report the performance on this test set. We use the macro- and micro-averages of F1-score as the performance measure in all following sections.

5.1 BoW & LSI

Performance of classifiers on different BoW models on the test set is shown in Table 6. Normalization seems beneficial for bigrams but not for unigrams, except for RF. Using bigrams and unigrams performs better than using only unigrams.

Additionally, the advantage of bigrams over just unigrams gets larger on larger data sets. For LogReg on Ekman, the absolute improvement is about 2 % while for POMS it is around 5 %.

When comparing different classifiers, RF always shows the worst performance while also requiring substantially more time. We speculate that the dimensionality of the space

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TAFFC.2018.2807817, IEEE Transactions on Affective Computing

TABLE 6
F1-scores (macro/micro) of different classifiers on different bag-of-words models as measured on the test set. For each of the averaging techniques, the best performance for each data set in either multiclass or multilabel approach is shown in bold.

		Multiclass				Multilabel			
		Unigrams		Bigrams		Unigrams		Bigrams	
		Vanilla	Norm.	Vanilla	Norm.	Vanilla	Norm.	Vanilla	Norm.
Ekman	SVM	58.5/69.4	58.1/69.5	59.5/70.6	**60.2**/71.2	55.5/66.8	54.8/66.7	56.4/68.2	56.4/69.2
	NB	56.0/67.3	56.4/67.1	55.6/68.1	56.2/68.4	55.1/65.8	55.4/65.7	55.2/66.7	55.8/66.8
	LogReg	58.7/69.5	58.7/69.7	59.2/70.8	60.1/**71.5**	55.5/67.1	55.7/67.3	56.3/68.5	**57.4**/**69.3**
	RF	46.5/62.3	47.2/63.5	45.8/62.1	46.5/63.1	35.8/53.7	35.7/54.2	36.2/55.2	35.9/55.8
Plutchik	SVM	54.4/66.8	54.1/66.7	56.3/68.2	**57.1**/69.0	50.3/63.2	49.9/62.9	52.5/65.3	**53.4**/66.3
	NB	51.8/64.5	52.2/64.3	52.5/65.6	53.0/66.0	50.4/62.6	51.0/62.5	50.8/63.1	51.4/63.3
	LogReg	54.7/67.0	54.8/66.9	56.0/68.5	57.0/**69.3**	51.3/63.9	51.0/63.8	52.7/65.5	53.3/**66.6**
	RF	41.9/59.1	42.9/60.4	42.0/59.4	42.7/60.6	28.4/46.9	29.1/48.3	28.9/48.3	29.6/49.8
POMS	SVM	63.8/66.8	63.5/66.8	67.5/71.0	68.2/71.5	59.0/61.4	58.4/61.1	65.1/67.9	65.9/68.5
	NB	61.0/64.4	60.9/64.4	63.9/66.9	64.2/67.3	58.3/62.7	58.3/62.6	60.3/65.5	60.5/65.8
	LogReg	64.2/67.0	64.0/67.0	68.1/71.2	**68.8**/**71.7**	60.3/62.6	60.0/62.4	65.3/68.3	**66.3**/**69.0**
	RF	57.6/64.0	59.1/65.2	57.2/63.9	59.1/65.3	44.4/51.5	46.0/53.0	47.3/56.5	49.1/57.7

TABLE 7
F1-scores (macro/micro) of different classifiers on different latent semantic indexing models as measured on the test set. For each of the averaging techniques, the best performance for each data set in either multiclass or multilabel approach is shown in bold.

		Multiclass				Multilabel			
		Unigrams		Bigrams		Unigrams		Bigrams	
		Vanilla	Norm.	Vanilla	Norm.	Vanilla	Norm.	Vanilla	Norm.
Ekman	SVM	40.5/57.9	39.2/57.0	53.4/66.9	53.2/67.0	33.3/49.5	31.5/47.8	48.2/63.2	47.8/63.2
	NB	31.8/41.0	32.7/42.2	22.5/37.0	24.7/38.1	33.4/39.6	34.4/41.4	27.9/33.2	29.0/34.5
	LogReg	42.2/58.5	40.8/57.5	54.5/67.0	**54.5**/**67.2**	35.7/51.6	33.9/49.9	**51.2**/64.3	51.1/**64.5**
	RF	36.6/55.0	37.7/56.1	30.9/51.8	33.7/53.6	24.0/34.8	26.2/37.7	14.1/22.7	16.1/25.6
Plutchik	SVM	34.0/53.8	32.7/52.7	49.5/63.9	49.1/64.0	25.1/42.8	23.1/40.6	42.5/58.8	41.8/58.7
	NB	25.9/35.4	26.6/36.3	16.7/30.4	18.5/31.3	26.8/30.1	27.3/31.4	23.7/26.7	24.6/27.8
	LogReg	36.4/54.4	34.5/53.3	50.8/64.2	**50.9**/**64.3**	29.0/45.7	26.6/43.5	**46.5**/60.5	46.1/**60.6**
	RF	31.5/52.0	31.9/52.5	27.3/49.1	29.3/50.8	18.7/31.1	20.1/33.5	9.9/17.9	12.2/21.2

is too large for RF, which has a rather small number of trees due to time limitations. Further, the performance of NB, SVM and LogReg is almost comparable, while LogReg seems slightly better than SVM, which in turn seems slightly better than NB. We observe that LogReg on normalized bigrams outperforms other combinations of models and classifiers. This indicates the usefulness of adding bigrams and normalizing the tokens.

We report the results for LSI models in Table 7. Some trends are similar to those observed for BoW. Normalization on bigrams seems to help most of the time, while on unigrams it is only beneficial for RF and NB. Contrary to BoW, adding bigrams only helps when using SVM or LogReg, but it hurts for NB and RF. The best overall result is again achieved by LogReg on normalized bigrams. Comparing to BoW, LSI models consistently show inferior performance. Hence, we will use LogReg on normalized bigrams of BoW as a baseline for comparison with NNs.

Finally, comparing different emotion classifications, we can conclude that POMS's categories can be predicted with the same accuracy as Ekman's: with our approach to labelling tweets, the higher complexity of POMS is counterbalanced by the larger number of useful tweets. The accuracy for Plutchik is a bit lower, which is a result of predicting two more classes comparing to Ekman or POMS.

5.2 RNNs & CNNs

Performance of neural networks is shown in Table 8. Here we summarize our main observations.

When exploring GloVe embedding, we confirm that it is too focused on words co-occurrences and that fine-tuning makes it more suitable for emotion detection. For example in GloVe, the most similar word to *sadness* is *happiness*; and to *angry* it is *birds* (in terms of cosine similarity). After fine-tuning, the closest word to *sadness* is *depression*; and to *angry* it is *annoyed*. Concluding from these concrete examples and the overall performance of models — which is always better with fine-tuning — the process of fine-tuning incorporates the notion of sentiment into the word embedding and makes them more useful for emotion prediction.

In seven out of eight cases we selected the largest embedding that GloVe provides (i.e. 200) while for one case the differences between 100- and 200-dimensional embeddings were negligible. Character embedding can indubitably be much smaller. Although we selected 25-dimensional representations due to slightly better results on the validation set, using just 10-dimensional ones decreases the performance by less than 1 % in absolute.

To evaluate the word embedding itself, we tested softmax on averaged word embeddings. Results are omitted for brevity, but they achieve micro averaged F1-scores below 60 %. Hence, recurrent or convolutional layer adds at least 10 % in absolute F1-score.

We observe that SimpleRNN layers consistently perform worse than LSTMs and GRUs, while the performances of the latter two are comparable. While multiple layers occasionally yield minor improvements, using bidirectional layers almost always helps when working on characters.

For CNNs, best kernel sizes are between 2–3 for words and 9–11 for characters. The number of feature maps is smaller for words (1500-2000) than for characters (5000-6000), but both ranges are larger than what was suggested by practitioners [31]. We conclude that more features are needed for larger data sets and for character approaches.

The success of character approaches is dependant on the data set size. For POMS, our largest data set, characters always beat word approaches; for Plutchik, second largest, in most cases, and for Ekman in some. Intuitively, with sufficient training data, NN works best directly on characters. Otherwise, it is best to rely on pre-trained embedding and fine-tune it further.

Comparing to BoW, RNNs almost always yielded better results while CNNs improved over BoW in most cases. These improvements come at the expense of higher computational costs. While training of LogReg on POMS data set took up to 13 hours, word-based RNNs could take up to 5 days and character-based ones up to 8 days on a single GPU. CNNs, due to simpler architecture, took from a few hours up to one day.

We recommend practitioners to opt for character-based recurrent neural networks. They showed as the most accurate, so whenever the performance is crucial, they should be preferred over bag-of-words or convolutional approaches. Further, they do not require any preprocessing, token normalization, stemming or filtering. Notice that without proper token normalization and without bigrams, bag-of-words approaches would lag behind neural approaches even more. Hence, since no language-specific prerequisites and no feature engineering is required, neural networks are more appropriate even for low-resource languages.

5.3 Transfer

Results of transfer experiments are summarized in Table 9. Note that for transfer experiments we first have to find a common architecture that works reasonably well for all three data sets. Hence, when comparing the results of transfer learning we should not compare them with results from Section 5.2, but rather with the performance on this common architecture, which is presented along each of the diagonals of Table 9.

We see that the only setting in which transfer learning gives comparable performance is when we train the initial model on Plutchik and then transfer it to Ekman: training the hidden layers on Plutchik gives even better performance than training on Ekman itself. Since Ekman is a subset of Plutchik, the examples corresponding to Ekman's categories are present in both settings. The only thing that differs is that the NN is trained with the presence of two additional Plutchik's categories *trust* and *anticipation*. As often observed in multitask learning, the signals from these two additional categories, although not directly useful, improve the generalization.

Transfer learning results in the other direction, i.e. from Ekman to Plutchik, are noticeably worse. The absence of *trust* and *anticipation* during training seems to have prevented the NN model from discovering the required features for their recognition. Furthermore, when inspecting the corresponding confusion matrices, we observe that *anticipation* is much more commonly confused with popular Ekman's categories (like *surprise*) comparing to what happens when training Plutchik itself on the common architecture.

Due to the similarity between Ekman and Plutchik, the real transfer scenario is to train either on Ekman or Plutchik and then test on POMS or the other way around. When training on Ekman and testing on POMS, the best performance we observe is about 58 %, while the worst just above 22 % micro F1-score. Comparing with the results of training POMS on a common architecture, which yield the accuracy of about 70 % micro F1-score, we observe that transfer capabilities of our models on such architecture are quite poor. Same holds for the transfer from Plutchik to POMS and for transfers from POMS to either Ekman or Plutchik. We observe that transfer capabilities of CNNs are in general better than on RNNs, which is understandable since our CNNs are much simpler, shallower architecture. Interestingly, the results are even worse for the multilabel setting than for multiclass one. These results suggest that when training on a single task, using final hidden state representation to predict other classifications will result in significant performance drops in comparison with training on the target classification.

5.4 Unison

While transfer shows inferior performance, we observe better results when training the network for all three targets together.

Performance by iterations for both strategies is depicted in Fig. 4. For the alternate batches strategy, we observe that the accuracy on the validation set for Ekman and Plutchik is almost flat and stops increasing through the last 15 epochs. Accuracy on the train set for Ekman and Plutchik is still rising, making the differences in accuracy larger and larger. This indicates that we have probably achieved the plateau on validation accuracy and that the model started to overfit. Contrary, the validation accuracy for POMS is slowly rising — almost in parallel with the train accuracy — throughout all iterations of the algorithm. The difference between the train and validation accuracies on POMS is much smaller than on Ekman or Plutchik even in the last iteration of the algorithm. We can assume that the accuracy for POMS would increase if the training continued. Notice that the average of all three validation accuracies has not improved in the last five iterations which triggered the early stopping. We believe that while the model had enough time to sufficiently train for Ekman and Plutchik, it got too few examples to train well for POMS. This discrepancy between the number of training examples the neural network must observe is exactly the issue that our newly proposed strategy is trying to solve.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TAFFC.2018.2807817, IEEE Transactions on Affective Computing

TABLE 8
F1-scores (macro/micro) for RNNs and CNNs as measured on the test test. For easier comparison we also show the best BoW performance. The best micro or macro score for each data set in either multiclass or multilabel setting is shown in bold.

		Multiclass			Multilabel		
	Input	RNN	CNN	*Best BoW*	RNN	CNN	*Best BoW*
Ekman	Word	**60.7**/71.4	59,9/69,2	*60.1/71.5*	57.3/70.0	**60,2**/69,8	*57.4/69.3*
	Char	60.4/71.2	60,1/**71,8**		56.4/**70.3**	54,9/68,9	
Plutchik	Word	52.8/65.5	55,1/66,8	*57.0/69.3*	55.5/67.7	**56,9**/67,0	*53.3/66.6*
	Char	**57.5**/**70.0**	57,0/69,2		52.6/**68.6**	53,9/66,8	
POMS	Word	67.9/70.4	67,1/69,9	*68.8/71.7*	67.0/70.4	65,7/68,8	*66.3/69.0*
	Char	**70.5**/**73.2**	68,1/70,9		**67.5**/**70.7**	66,7/69,0	

TABLE 9
Transfer experiments F1-scores (macro/micro) for word- and character-approaches in multiclass and multilabel settings on the test set. The table consists of eight 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.

				Softmax Tuning & Testing					
				Multiclass			Multilabel		
				Ekman	Plutchik	POMS	Ekman	Plutchik	POMS
Initial Weight Learning	RNN	Word	Ekman	57.9/68.9	44.9/59.6	37.1/49.1	56.9/69.6	37.0/47.3	9.8/22.1
			Plutchik	57.7/69.1	54.1/66.0	37.3/49.3	52.8/67.6	51.0/66.5	11.5/20.3
			POMS	51.5/63.0	48.0/59.7	68.5/71.3	6.1/9.4	3.1/4.1	67.0/70.4
		Char	Ekman	60.3/71.8	46.5/60.5	36.5/49.2	55.0/70.7	40.3/54.0	20.3/30.4
			Plutchik	61.1/72.4	58.5/69.9	39.5/50.9	55.7/70.5	52.8/68.3	23.4/33.2
			POMS	37.6/51.1	31.9/45.5	69.9/72.8	21.5/32.7	12.0/16.6	67.5/70.5
	CNN	Word	Ekman	59.9/70.1	51.7/62.7	52.9/57.8	57.8/69.2	51.0/63.6	51.9/55.5
			Plutchik	59.3/69.4	56.0/66.9	53.6/57.9	59.3/69.5	55.5/67.3	50.9/55.8
			POMS	45.6/56.3	39.1/51.2	66.7/69.8	44.4/57.1	39.6/51.6	65.9/68.3
		Char	Ekman	57.4/69.6	50.0/64.0	48.7/58.0	59.1/70.1	51.5/64.2	51.8/56.1
			Plutchik	60.0/71.5	54.3/68.0	50.5/59.2	56.4/69.7	55.5/67.0	51.7/56.9
			POMS	49.3/62.7	44.8/58.2	65.8/69.5	49.2/61.5	43.4/55.9	64.4/66.7

When using weighted sampling batches strategy, we observe the following. First, and this holds for all eight runs, the training process requires many more iterations. When the training stops, the difference between train and validation accuracies for Ekman and Plutchik is considerably smaller than when using alternate batches strategy. Contrary, and as expected, for POMS this difference becomes larger. Sampling probabilities through iterations of train-validation setting depicted in Fig. 5 show that when using weighted sampling many more training instances came from POMS data set. After some oscillations in the first few iterations, the sampling probability for POMS converged to approximately 0.7, while probabilities for Ekman and Plutchik are around 0.14 and 0.16. Assuring that model sees more than twice as many examples from POMS than of Ekman and Plutchik combined, enabled improving the performance of POMS while not harming Ekman and Plutchik.

The real assessment of our heuristic is its performance on the test set, which is shown in Table 10. For Ekman and Plutchik, the alternate batches strategy works comparable, usually within 1 % absolute deviation, to the performance of single models. Contrary, the performance on POMS is degraded by about 5 % in absolute. This confirms our suspicion that the alternate batches strategy does not train enough on POMS which causes inferior performance. By using weighted sampling instead, we managed to improve on the results of POMS and make its performance comparable with that of a single model, while also not wrecking the performance of Ekman and Plutchik.

We observe that, except for RNNs with words on input in multiclass setting, WSB always outperforms WSBDS. Further, comparing WSB with AB, WSB outperforms AB in the great majority of the cases. We must emphasize that in terms of the harmonic average, the best performance across all three data sets in either multiclass or multilabel setting was always achieved with the WSB strategy. This indicates that neither simply iterating nor weighting by data set size does not yield the optimal performance; hence a smarter heuristic is required when the data sets differ in size or complexity. Finally, using weighted sampling brings us closer to the average performance of three single models while restricting ourselves to use a single model instead.

5.5 Limitations & Future Work

Most of study's limitations stem from the creation of the data sets using distant supervision. The most obvious one is the class imbalance problem. Observing the confusion matrices — one is shown in Fig. 6 — revealed that the models are biased towards the categories with large number of examples (e.g. *joy*), while they perform quite poorly for categories with very little examples (e.g. *disgust*). To

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TAFFC.2018.2807817, IEEE Transactions on Affective Computing

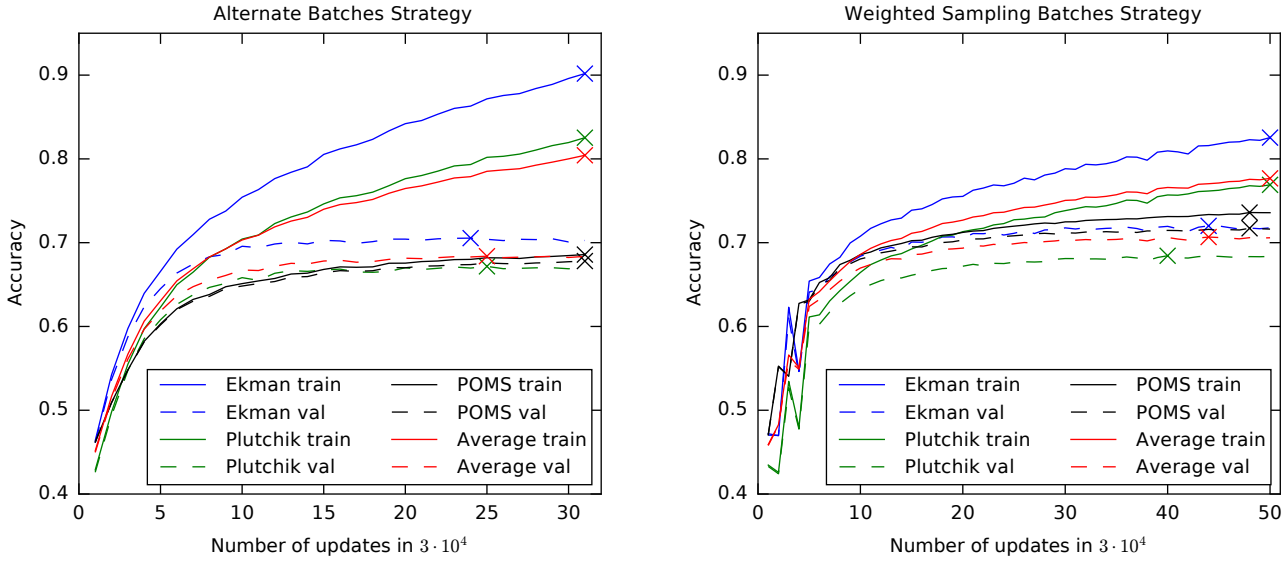

Fig. 4. Comparison of accuracies through iterations of training unison model for alternate batches (left) and weighted sampling batches strategies (right) in train-validation setting. Cross marks indicate on the maximum of each curve. Sampling probabilities corresponding to the weighted sampling batches strategy are shown in Fig. 5. We are showing a comparison on characters in multiclass setting using RNNs while the same trends are present in other settings as well.

TABLE 10
F1-scores (macro/micro) of unison model on the test set when trained with all three strategies. In each setting, the higher score between alternate batches (AB), weighted sampling batches (WSB) and weighted sampling batches according to data set sizes (WSBDS) strategy is shown in bold.

NN	Input	Strategy	Multiclass				Multilabel			
			Ekman	Plutchik	POMS	Harmonic	Ekman	Plutchik	POMS	Harmonic
RNN	Word	AB	**61.1**/**71.5**	56.7/68.5	64.3/67.7	60.5/69.2	**58.2**/**71.5**	**54.0**/**68.0**	62.2/65.9	**57.9**/68.4
		WSBDS	59.6/71.3	56.6/68.2	**69.1**/**71.7**	61.3/**70.4**	52.8/68.5	49.4/65.1	**67.2**/**69.9**	55.5/67.8
		WSB	60.3/71.0	**57.5**/**68.6**	68.2/71.1	**61.7**/70.2	55.1/70.1	51.2/66.8	65.8/69.0	56.7/**68.6**
	Char	AB	60.4/71.9	56.7/68.6	63.8/68.0	60.2/69.5	**56.3**/**71.0**	**52.7**/**67.3**	60.2/65.0	56.2/67.7
		WSBDS	58.7/71.0	54.6/67.8	69.4/**72.3**	60.3/70.3	54.3/69.6	49.4/65.9	**67.8**/**70.9**	56.2/68.8
		WSB	**61.8**/**73.0**	**57.8**/**69.5**	**69.5**/72.2	**62.7**/**71.5**	55.1/70.4	52.3/**67.3**	66.2/69.7	**57.3**/**69.1**
CNN	Word	AB	**59.2**/**69.7**	**54.7**/**66.7**	62.9/66.2	58.7/67.5	**57.2**/**69.2**	**53.6**/**66.4**	62.1/65.4	57.4/**67.0**
		WSBDS	57.6/68.2	53.6/65.0	**66.2**/**69.2**	58.7/67.4	53.1/66.8	48.0/64.0	**65.3**/**67.7**	54.6/66.2
		WSB	57.8/69.1	54.4/65.8	65.6/68.3	**58.9**/**67.7**	56.6/68.2	53.3/65.0	64.8/67.0	**57.9**/66.7
	Char	AB	**60.3**/**71.7**	**55.2**/**68.1**	61.8/66.5	**59.0**/**68.7**	59.9/69.3	53.4/66.9	61.6/63.4	58.1/66.5
		WSBDS	56.6/69.1	52.3/65.5	**65.3**/**69.0**	57.6/67.8	57.5/68.7	52.8/65.1	**65.9**/**67.9**	58.2/67.2
		WSB	57.8/69.7	52.5/66.0	64.8/68.5	57.9/68.1	**60.4**/**70.1**	**54.0**/**67.1**	65.8/67.6	**59.7**/**68.3**

alleviate this, one could incorporate morphological variations, synonyms or derivatives of words when searching for emotional hashtags. For example, instead of only searching for tweets with a hashtag #sadness one could also search for #sad, #unhappy or #sosad. Also, the neutral class should be added to properly distinguish between emotional and factual tweets. Since there were no humans involved, we could not properly address sarcastic tweets. Hence, at least the category *joy* contains some of them. While the classifier might learn to put joyful and sarcastic tweets into the category *joy*, this is problematic for interpretation. Finally, before putting the classifiers into practical use, an evaluation on standardized, unfiltered, preferably human-annotated set of tweets would be crucial. We know the models can distinguish between emotions, but their performance in the wild is an open question.

Another interesting idea would be to study the inter-dependence between emotion classifications and to test the generality of unison model's final hidden state representations for predicting other classifications or sentiment.

6 Related Work

We describe the related work on emotion prediction in Section 6.1 and the work on training approaches in Section 6.2.

6.1 Related Work on Emotion Prediction

The first prediction models for Ekman's six basic emotions date back at least a decade. Alm *et al.* [9] 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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TAFFC.2018.2807817, IEEE Transactions on Affective Computing

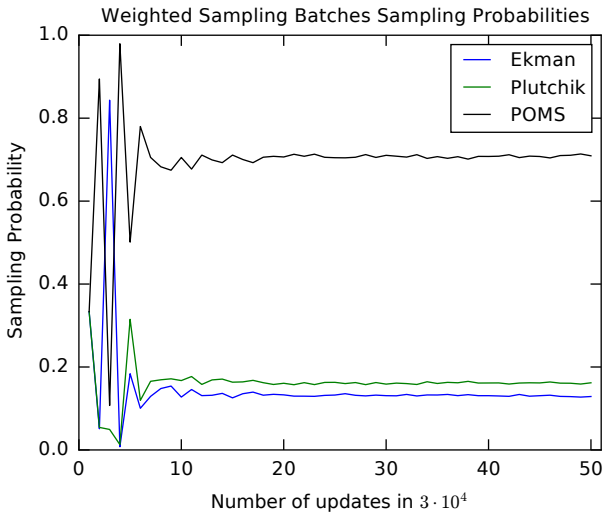

Fig. 5. Sampling probabilities of weighted sampling batches strategy corresponding to the accuracy plot shown in Fig. 4. These probabilities were estimated in multiclass setting using characters as input to RNNs while plots for other settings exhibit the same trends.

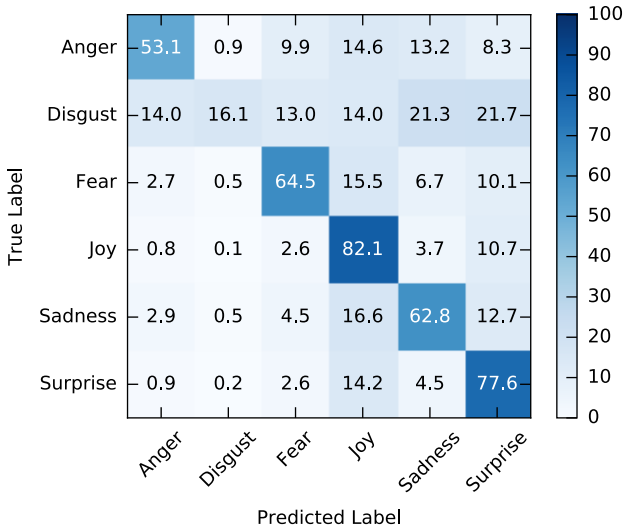

Fig. 6. Confusion matrix on the Ekman's test data for unison model trained using weighted sampling batches strategy. Each cell shows the percentage of corresponding examples while the percent signs are omitted due to brevity.

a corpus of blog posts but again distinguished only between classes *emotion* and *no emotion*. In 2007, SemEval held a competition in emotion detection 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 the emotion labelling, while others only worked on the polarity classification. The emotion labelling ones were a rule-based system using lexicons, a system exploiting Point-wise Mutual Information (PMI) scores gathered through three different search engines, and a supervised system using unigrams. Their averaged F1-scores over emotion categories were around 10 %. Performance on this data was later improved to an 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 optimization SVM yields the greatest improvement over simple baselines. The closest to our approach is the work of Mohammad & Kiritchenko [7] who exploited hashtags corresponding to Ekman's emotion categories to obtain a labeled data sets of 21,051 tweets. With cross-validating an SVM on n-grams they obtain a micro-averaged F1-score of 49.9 %. Comparing this to our best overall result for Ekman (73.0 % in unison model) and to the best bag-of-words models (71.5 %) we believe that most of this difference can be contributed to the approximately twenty times larger data set.

Work on Plutchik's emotions includes Mohammad & Turney, who created an emotion lexicon using Amazon's Mechanical Turk [36]. Later Mohammad *et al.* [8] collected a set of about 2000 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 [37] 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. We improve over both of these results to a 70.0 % F1-score, which again, we mostly contribute to the significantly larger data set.

The work on POMS is quite rare, as the test is available only to professional psychologists. Most existing studies were led by Johan Bollen. Common to all is the idea of tracking adjectives defined in the POMS questionnaire and using its structure to obtain six-dimensional mood representation. Bollen investigated how Twitter mood predicts the stock market changes [1], [38]. In a similar study [39], he correlated emotion time series with records of popular events and showed that such events may have a significant effect on various dimensions of public mood. By analyzing emails submitted to *futureme.org*, Pepe & Bollen also revealed the long-term optimism of its users, but medium-term confusion [40]. Those studies used POMS questionnaire as a tool for obtaining mood representations but did not study the problem of predicting POMS's categories from the text.

There are several studies that use other categorizations of emotions. Neviarouskaya and colleagues developed two rule-based systems for detecting nine Izard emotions; one works on blogs [41], another one on personal stories from experience project[11] website [42]. Mishne [43] experimented with detecting 40 different mood states on blog posts from the LiveJournal community. He used features related to n-grams, length, semantic orientation of words, PMI, emphasized words and special symbols to train an SVM classifier. Mihalcea & Liu [44] used a subset of these blog posts to train a Naïve Bayes classifier for distinguishing between *happy* and *sad* posts. Yerva *et al.* [45] fused weather-dependent mood representations from Twitter with real-time meteo-

11. http://www.experienceproject.com

rological data to provide travel recommendations based on the expected mood of people in a particular city.

Although we approach the problem by essentially predicting hashtags, our study differs from the usual hashtag recommendation [46], [47], [48] in that those studies usually choose among tens of thousands of different hashtags but with potentially similar meanings, while we target a small set of hashtags corresponding to distinct emotions.

6.2 Related Work on Training Heuristics

The alternate batches heuristic was introduced by Collobert & Weston [30] and has later been used in many other studies [49], [50], [51]. We improve upon this heuristic in cases where the tasks differ in complexity or data set sizes.

Our setting also resembles the multimodal learning approaches [52]; however, we worked with three different data sets and not a single data set containing instances described with multiple modalities. Further, the loss aggregating approaches, like [53], are not directly applicable since they optimize the similarity between hidden state representations of different modalities while we insist on having a common projection of tweets into the final hidden state for all three emotion classifications. Hence, we choose the training heuristic by Collobert & Weston [30] since it has already proven successful for NLP.

7 Conclusion

The central aim of the paper was to explore the use of deep learning for emotion detection. We created three large collections of tweets labeled with Ekman's, Plutchik's and POMS's classifications of emotions. Recurrent neural networks indeed outperform the baseline set by the common bag-of-words models. Our experiments suggest that it is better to train RNNs on sequences of characters than on sequences of words. Beside more accurate results, such approach also requires no preprocessing or tokenization. We discovered that transfer capabilities of our models were poor, which led us to the development of single unison model able to predict all three emotion classifications at once. We showed that when training such model, instead of simply alternating over the data sets it is better to sample training instances weighted by the progress of training. We proposed an alternative training strategy that samples training instances based on the difference between train and validation accuracy and showed that it improves over alternating strategy. We confirmed that it is possible to train a single model for predicting all three emotion classifications whose performance is comparable to the three separate models. As a first study working on predicting POMS's categories, we believe they are as predictable as Ekman's and Plutchik's. We also showed that searching for tweets containing POMS adjectives and later grouping them according to POMS factor structure yields a coherent data set whose labels can be predicted with the same accuracy as other classifications.

We made our character-based trained RNN models publicly available at https://github.com/nikicc/twitter-emotion-recognition.

We worked on probably the largest data set for emotion prediction, using tweets from the last seven years. With the aim of developing a universal emotion detection algorithm, we did not restrict ourselves only to one domain, but rather tested its usefulness for different classifications of emotions. Since the training data was annotated automatically and since we use character-based approaches, our solution is language independent and could easily be adapted for other languages.

Past studies of this problem focused on somewhat different goals and used much smaller collections of tweets, which prevented the use of deep learning and resulted in discouragingly low classification performance [7], [33], [34]. Our study, however, shows that, given enough data, emotion prediction may not be such a hard problem after all.

References

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

[2] D. Gruhl, R. Guha, R. Kumar, J. Novak, and A. Tomkins, "The Predictive Power of Online Chatter," *Proc. of the 11th ACM SIGKDD Int. Conf. on Knowledge discovery in data mining*, pp. 78–87, 2005.

[3] G. Mishne and N. Glance, "Predicting Movie Sales from Blogger Sentiment," *AAAI Spring Symposium: Computational Approaches to Analyzing Weblogs*, pp. 155–158, 2005.

[4] P. Ekman, "An Argument for Basic Emotions," *Cognition & Emotion*, vol. 6, no. 3, pp. 169–200, 1992.

[5] R. Plutchik, "A General Psychoevolutionary Theory of Emotion," in *Theories of Emotion*. Academic Press, 1980, vol. 1, pp. 3–33.

[6] J. C. Norcross, E. Guadagnoli, and J. O. Prochaska, "Factor structure of the Profile Of Mood States (POMS): Two Partial Replications," *J. of Clinical Psychology*, vol. 40, no. 5, pp. 1270–1277, 1984.

[7] S. M. Mohammad and S. Kiritchenko, "Using Hashtags to Capture Fine Emotion Categories from Tweets," *Computational Intelligence*, vol. 31, no. 2, pp. 301–326, 2015.

[8] S. M. Mohammad, X. Zhu, S. Kiritchenko, and J. Martin, "Sentiment, emotion, purpose, and style in electoral tweets," *Information Processing and Management*, vol. 51, no. 4, pp. 480–499, 2015.

[9] C. O. Alm, D. Roth, and R. Sproat, "Emotions from text: machine learning for text-based emotion prediction," in *Proc. of Human Language Technology Conf. and Conf. on Empirical Methods in Natural Language Processing*, no. October. ACL, 2005, pp. 579–586.

[10] B. Plank and D. Hovy, "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*, 2015, pp. 92–98.

[11] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank," in *Proc. of the Conf. on Empirical Methods in Natural Language Processing*. Citeseer, 2013, pp. 1631–1642.

[12] O. Irsoy and C. Cardie, "Opinion Mining with Deep Recurrent Neural Networks," in *Proc. of the Conf. on Empirical Methods in Natural Language Processing*. ACL, 2014, pp. 720–728.

[13] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural Language Processing (almost) from Scratch," *J. of Machine Learning Research*, vol. 12, pp. 2493–2537, 2011.

[14] A. Radford, R. Jozefowicz, and I. Sutskever, "Learning to Generate Reviews and Discovering Sentiment," *arXiv preprint arXiv:1704.01444*, 2017.

[15] A. Go, R. Bhayani, and L. Huang, "Twitter Sentiment Classification using Distant Supervision," *CS224N Project Report, Stanford*, vol. 1, pp. 1–6, 2009.

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

[17] E. Kouloumpis, T. Wilson, and J. Moore, "Twitter Sentiment Analysis: The Good the Bad and the OMG!" *Proc. of the 5th Int. AAAI Conf. on Weblogs and Social Media (ICWSM 11)*, pp. 538–541, 2011.

[18] R. González-Ibáñez, S. Muresan, and N. Wacholder, "Identifying Sarcasm in Twitter: A Closer Look," in *Proc. of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, no. 2010. ACL, 2011, pp. 581–586.

[19] D. Bamman and N. A. Smith, "Contextualized Sarcasm Detection on Twitter," in *Proc. of the 9th Int. AAAI Conf. on Web and Social Media*. Citeseer, 2015, pp. 574–577.

[20] K. Gimpel, N. Schneider, B. O'Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yogatama, J. Flanigan, and N. A. Smith, "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*, vol. 2, no. 2. ACL, 2011, pp. 42–47.

[21] I. T. Jolliffe, "Principal Component Analysis," in *Springer Series in Statistics*, 2002.

[22] R. B. Bradford, "An Empirical Study of Required Dimensionality for Large-scale Latent Semantic Indexing Applications," *Proc.of the 17th ACM Conf.*, p. 153, 2008.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *J. of Machine Learning Research*, vol. 12, pp. 2825–2830, 2012.

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

[25] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global Vectors for Word Representation," in *Proc. of the 2014 Conf. on Empirical Methods in Natural Language Processing*. ACL, 2014, pp. 1532–1543.

[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," in *J. of Machine Learning Research*, vol. 15, 2014, pp. 1929–1958.

[27] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent Neural Network Regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[28] F. Chollet, "Keras," https://github.com/fchollet/keras, 2015.

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

[30] R. Collobert and J. Weston, "A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning," in *Proc. of the 25th Int. Conf. on Machine Learning*, vol. 20, no. 1. ACM, 2008, pp. 160–167.

[31] Y. Zhang and B. C. Wallace, "A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification," *ArXiv preprint arXiv:1510.03820v4*, 2016.

[32] S. Aman and S. Szpakowicz, "Identifying Expressions of Emotion in Text," in *Int. Conf. on Text, Speech and Dialogue*, vol. 4629. Springer, 2007, pp. 196–205.

[33] C. Strapparava and R. Mihalcea, "SemEval-2007 Task 14: Affective Text," in *Proc. of the 4th Int. Workshop on Semantic Evaluations*. ACL, 2007, pp. 70–74.

[34] ——, "Learning to Identify Emotions in Text," in *Proc. of the 2008 ACM symposium on Applied Computing*. ACL, 2008, pp. 1556–1560.

[35] S. Chaffar and D. Inkpen, "Using a Heterogeneous Dataset for Emotion Analysis in Text," in *Canadian Conf. on Artificial Intelligence*. Springer, 2011, pp. 62–67.

[36] S. M. Mohammad and P. D. Turney, "Emotions Evoked by Common Words and Phrases: Using Mechanical Turk to Create an Emotion Lexicon," in *Proc. of the NAACL HLT 2010 Workshop on Computational Approaches to Analysis and Generation of Emotion in Text*. ACL, 2010, pp. 26–34.

[37] E. Tromp and M. Pechenizkiy, "Rule-based Emotion Detection on Social Media: Putting Tweets on Plutchik's Wheel," *arXiv preprint arXiv:1412.4682*, 2014.

[38] J. Bollen and H. Mao, "Twitter Mood as a Stock Market Predictor," *IEEE Computer Society*, vol. 44, no. 10, pp. 91–94, 2011.

[39] J. Bollen, H. Mao, and A. Pepe, "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*, 2011, pp. 450–453.

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

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

[42] ——, "@AM: Textual Attitude Analysis Model," in *Proc. of the NAACL HLT 2010 Workshop on Computational Approaches to Analysis and Generation of Emotion in Text*. ACL, 2010, pp. 80–88.

[43] G. Mishne, "Experiments with Mood Classification in Blog Posts," in *Proc. of ACM SIGIR 2005 Workshop on Stylistic Analysis of Text for Information Access*, vol. 19, 2005, pp. 321–327.

[44] R. Mihalcea and H. Liu, "A Corpus-based Approach to Finding Happiness," in *AAAI Spring Symposium: Computational Approaches to Analyzing Weblogs*, 2006, pp. 139–144.

[45] S. R. Yerva, J. Hoyoung, and K. Aberer, "Cloud based Social and Sensor Data Fusion," in *15th Int. Conf. on Information Fusion*. IEEE, 2012, pp. 2494–2501.

[46] J. Weston, S. Chopra, and K. Adams, "#TagSpace: Semantic Embeddings from Hashtags," in *Proc. of the 2014 Conf. on Empirical Methods in Natural Language Processing*. Citeseer, 2014, pp. 1822–1827.

[47] S. M. Kywe, T.-A. Hoang, E.-P. Lim, and F. Zhu, "On Recommending Hashtags in Twitter Networks," in *Int. Conf. on Social Informatics*. Springer, 2012, pp. 337–350.

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

[49] J. Guo, W. Che, H. Wang, and T. Liu, "A Universal Framework for Inductive Transfer Parsing across Multi-typed Treebanks," *Proc. of the 26th Int. Conf. on Computational Linguistics (COLING-16)*, pp. 12–22, 2016.

[50] B. Nejat, G. Carenini, and R. Ng, "Exploring Joint Neural Model for Sentence Level Discourse Parsing and Sentiment Analysis," *Proc. of the SIGDIAL 2017 Conf.*, no. August, pp. 289–298, 2017.

[51] X. Liu, J. Gao, X. He, L. Deng, K. Duh, and Y.-Y. Wang, "Representation Learning Using Multi-Task Deep Neural Networks for Semantic Classification and Information Retrieval," *Proc.of the 2015 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 912–921, 2015.

[52] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal Deep Learning," *Proc. of the 28th Int. Conf. on Machine Learning (ICML)*, pp. 689–696, 2011.

[53] F. Feng, X. Wang, and R. Li, "Cross-modal Retrieval with Correspondence Autoencoder," *Proc.of the 22nd ACM Int. Conf. on Multimedia*, pp. 7–16, 2014.

Niko Colnerič received the BSc degree in computer science from the University of Ljubljana, Slovenia in 2013. He is currently pursuing the PhD degree in the field of machine learning at University of Ljubljana. His research interests include text mining, opinion mining and emotion recognition from large unstructured data sets, especially the web.

In 2014 he received an ASEF fellowship for summer research visit to the Stanford University, which initiated the work presented by this paper.

Janez Demšar received his PhD at the University of Ljubljana, where he teaches courses in programming, algorithms and didactics. His research interests include machine learning, statistics and visualization, and he is one of the core developers of data mining suite Orange. He also often organizes various outreach activities and occasionally teaches computer science in primary schools.