Exploiting Deep Neural Networks for Tweet-based Emoji Prediction

Andrei Catalin Coman[1], Giacomo Zara[1], Yaroslav Nechaev[2], Gianni Barlacchi[2], and Alessandro Moschitti[1]

[1] University of Trento, Trento, Italy
{andreicatalin.coman, giacomo.zara}@studenti.unitn.it,
moschitti@disi.unitn.it
[2] Fondazione Bruno Kessler, Trento, Italy
{nechaev, barlacchi}@fbk.eu

Abstract. For many years now, emojis have been used in social networks and chat services in order to enrich written text with auxiliary graphical content, achieving a higher degree of empathy. In particular, given the wide use of this medium, emojis are now available in such a number and variety that every basic concept and mood is covered by at least one. For this reason, the connection between the emoji and its semantical meaning grows stronger. In this paper, we will be describing the work performed in order to develop a Machine Learning based tool that, given a tweet, predicts the most likely emoji associated with the text. The task resembles the one presented by Barbieri et al., [23], and is placed within the context of the International Workshop on Semantic Evaluation (SemEval) 2018. We designed a baseline with standard Support Vector Machines and another baseline based on fastText, which was provided as part of the Workshop. In addition, we implemented several models based on Neural Networks such as Bidirectional Long Short-Term Memory Recurrent Neural Networks and Convolutional Neural Networks. We found that the latter is the most effective since it outperformed all our models and ranks in the 6th position out of 47 total participants. Our work aims to illustrate the potential of simpler models, which, thanks to the fine-tuning of hyper-parameters, could achieve accuracy comparable to the more complex models of the challenge.

Keywords: emoji · Twitter · Text Classification · Machine Learning · SVM · fastText · Bi-LSTM · CNN

1 Introduction

We develop a Machine Learning model, which, given the text of a tweet, predicts the most likely associated emoji among the most common 20 shown in Table 1. In order to do this, we have first implemented a **baseline** approach based on Support Vector Machine (SVM), and used a `fastText`[3] classifier as an alternative baseline. After the implementation of the baselines, we proceeded to the

[3] **fastText**: https://github.com/facebookresearch/fastText

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
❤️	😍	😂	💕	🔥	😊	😎	✨	💙	😘	📷	🇺🇸	☀️	💜	😉	💯	😄	🎄	📸	😜

Table 1: The 20 most common emojis used as labels

development of **Neural Networks** based classifiers. In particular, we have chosen to test two different models: a **Bidirectional Long Short-Term Memory** (Bi-LSTM) Recurrent Neural Network (RNN), and a **Convolutional Neural Network** (CNN). The objective was to verify whether we would manage to outperform the results obtained by the baselines, and which of the two models would work better. For the whole task, we used the data provided within the **International Workshop on Semantic Evaluation 2018**[4][1].

Given the wide use of emojis that we are witnessing nowadays, not just on Twitter[5] but on many other Social Media and Instant Messaging services [12], one of the purposes of our paper consists in studying the relation between the text written by the user and the emojis related to it, through the implementation of a predicting tool that attempts to mimic the reasoning according to which a certain emoji is **evoked** by the written message. This turns out to be quite a complex problem since it is highly dependent on the understanding that human beings have of emojis, which is hardly ever the same. This ambiguity is what most strongly limits the effectiveness of a prediction task performed on this domain. In fact, the results obtained by the top-ranked participants as well as ours, which are reported in Section 6 together with a discussion, show how difficult such a task can be. Our work, however, aims to be an alternative to the more complex models used by the other participants, with the objective of demonstrating that even simpler models are able to easily overcome the baselines and compete with the more performing ones, thanks to the fine-tuning of hyper-parameters.

2 Related work

Obviously, a significant amount of work has been performed in the scope of the SemEval competition, by the participants of its past editions. In particular, it is worth to mention the work of Çöltekin and Rama [8], who won the competition by means of an SVM classifier. In their feature extraction procedure, they took into account both the character-level and word-level in order to extract the bag-of-n-grams features. Follows the work done by Baziotis et Al. [4], who used a Bidirectional LSTM (Bi-LSTM) with attention, and pre-trained word2vec vectors, obtaining the highest recall. The third place is occupied by the user `hgsgnlp` who has not provided any publication of his work and we send to [1] for the original reference. In the next position we find Liu [19], who instead,

[4] **SemEval**: `http://alt.qcri.org/semeval2018/`
[5] **Twitter**: `http://twitter.com/`

proposed a Gradient Boosting Regression Tree approach combined with a Bi-LSTM on character and word n-grams. Fifth position is occupied by Lu et. Al [20] who adopted a Bi-LSTM with attention and used different embeddings such as word, char, Part of Speech (POS) and Named Entity Recognition (NER) embeddings together with Twitter specific features like punctuation, all-caps and bag-of-hashtags. After them follows the work of Beaulieu and Owusu [5] who combined SVM with bag-of-words approach, obtaining a very high precision. The comparison between the above mentioned top 6 teams and our work can be seen in Table 4. Please refer to [1] for the full table of results of all participants.

Among the other participants it is worth mentioning Coster et Al. [10] who built an SVM classifier on characters and word n-grams, obtaining the second best results in the Spanish task. Basile and Lino [3] applied an SVM over a variety of features (such as *tf-idf*, used in this paper too, POS tags and bigrams), obtaining a competitive system. Finally, Jin and Pedersen [15] applied a soft voting ensemble approach combining different Machine Learning algorithms such as Multinomial Naive Bayes, Logistic Regression and Random Forests.

Beside sequence-to-sequence models, another fast and performing approach to solve NLP is given by the use of CNNs. These models demonstrated to achieve state-of-the-art performances in many NLP tasks like sentence classification [17] and answer reranking [25]. For instance, Zhao and Zeng [29] obtained better results in emoji prediction with Twitter data by applying CNNs. A related task has also been performed by Felbo et Al [11], who extended supervised learning to a set of labels represented by emojis themselves, in the scope of sentiment analysis. Similarly, Wolny [28] has extended binary sentiment analysis to multi-class by means of emojis. Barbieri et Al. [2] performed a study on emojis, with particular respect to their usage through time.

3 Problem statement and baseline

As previously anticipated, the task faced in this project consisted of developing a Machine Learning based prediction framework which, given the text of a tweet, is able to predict the most likely associated emoji among the most common 20. In order to train our models, we have been using the dataset provided for the SemEval competition. This dataset has been obtained by selecting only tweets that contain **exactly one** emoji, removing such emoji from the text and using it as **label**.

The **training set** consists of 475.624 labeled tweets, for a total of about 330.000 unique tokens. This is the dataset that we have been using to train both the baseline and the Neural Network models. The **test set**, instead, consists of 50.000 labeled tweets, for a total of about 70.000 unique tokens, resembling the distribution of the ones in the training set.

3.1 Support Vector Machine baseline

As mentioned in the introduction, one of the baselines has been implemented as an SVM-based classifier. The whole **preprocessing** procedure has been car-

ried out by means of the functionalities provided by `Apache Spark`[6], an analytic engine for data processing based on distributed computation. For the **classification** phase, we relied on `LIBLINEAR`[7], a library for large-scale linear classification.

In order to be fed to the classifier, our data have firstly been processed. In particular, the text has been set to **lowercase**, and then **tokenized** using the `NLTK`[8] module for `Python`[9]. Once having done this, each tweet had to be turned into a vector, in order for it to be classified. For this purpose, we have decided to apply *tf-idf* (Term-Frequency Inverse-Document-Frequency) analysis. The result of the *tf-idf* **vectorization** is a $m \times n$ matrix M, where m is the number of tweets, and n is the number of distinct words in the whole dataset. The cell $M[i, j]$ contains the *tf-idf* value of the j-th term with respect to the i-th tweet. Such value is formally defined as follows:

$$tf_{ij} = \frac{n_{ij}}{|d_i|} \tag{1}$$

$$idf_j = \log \frac{|D|}{|\{d : j \in d\}|} \tag{2}$$

$$tfidf_{ij} = tf_{ij} \times idf_j \tag{3}$$

where n_{ij} is the number of occurrences of term j in document i, $|d_i|$ is the size, in terms of words, of document i, $|D|$ is the size, in terms of documents, of the whole dataset, and $|\{d : j \in d\}|$ is the number of documents in the dataset which contain j. This is based on the idea that a word is most likely to be meaningful for a tweet if it appears as often as possible in the tweet, and as few times as possible in other tweets.

The vectorization has been performed applying different ways to extract n-grams from the text, in terms of type and strategy. In particular, we have been iterating over different parameters in terms of n-ngram size, n-gram type, presence/absence of punctuation and minimum document frequency. The n-gram size simply indicates how many single words are to be included in each feature, while the n-gram type indicates whether the n-gram consists of a sequence of **consecutive words** or if it involves **skips**.

3.2 fastText baseline

FastText is an open-source library that allows users to learn **unsupervised text representations** [6] and perform **supervised text classifications** [16].

In terms of **word representations**, this library follows a very similar approach to the one proposed in [21]. Unlike *word2vec*, where each word is seen as a single unit to which one must associate a vector, `fastText` considers a single word as a character-level n-gram composition, where each n-gram has its own

[6] **Apache Spark**: `https://spark.apache.org/`
[7] **LIBLINEAR**: `https://www.csie.ntu.edu.tw/~cjlin/liblinear/`
[8] **NLTK**: `https://www.nltk.org/`
[9] **Python**: `https://www.python.org/`

representation, which must be learned. This allows, for example, rare words to still have n-grams shared with other words. The same concept is applicable to out-of-vocabulary words, i.e., words that are present in the test set but are missing in the train set. The representation of a missing word as a concatenation of the vectors of the n-grams of which it is composed can solve this problem considering that presumably the representation of each n-gram has already been learned at an earlier stage. Concerning the **classification** part, a $softmax$ function is used to compute the probability distribution over the predefined classes. For a set of T tweets, this leads to minimizing the negative log-likelihood over the classes(emojis):

$$-\frac{1}{T}\sum_{t=1}^{T} e_t \log(softmax(BAx_t)) \tag{4}$$

where the first weight matrix A is a look-up table over the words, B is the weight matrix learned during the training phase, e_t represents the class(emoji) of the corresponding tweet, and x_t is the normalized bag of features of the t-th tweet. The latter vector can be seen as the average of the n-grams embeddings that form the corresponding tweet.

4 Deep learning models

In this Section we present the Neural Network models we used to perform our emoji predictions. The first model is an architecture based on RNNs [14], where the recurrent cell consists of a Bi-LSTM [13,24]. As second model we decided to implement a CNN [18], since over the years it has proved to be effective in the Natural Language Processing area [9]. Some other concepts are also exposed including the last layer used for predictions, the loss function applied for the train, the regularization techniques employed to contain overfitting and finally the strategy adopted to stop the train.

4.1 Bidirectional Long Short-Term Memory Recurrent Neural Networks

Over the years, RNNs have proven to be very effective in tasks that involve data in the form of sequences. With the advent of LSTMs, they have become even more powerful, as they have enabled us to tackle the problem of long-term dependencies. They are capable of remembering information for long periods of time. To do this, they maintain a state that is updated each time new information is being examined. Information update can be seen as the transition from one state, or rather from one **cell state** to another. The addition or removal of information to the cell state is regulated by structures called **gates**, where each has a specific task. These structures can be described more formally by means of the following formulae[10]:

[10] **Understanding LSTM Networks**: http://colah.github.io/posts/2015-08-Understanding-LSTMs

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{5}$$

which denotes the **forget gate layer**. Here we decide which information we are going to remove from the previous cell state. In order to take this decision we look at the output h_{t-1} of the previous cell and the current input x_t, which in our case consists of a token. The output of the sigmoid function σ is a vector of numbers between `0` and `1` for each element in the cell state C_{t-1}. This number indicates the importance of that piece of information. A value close to zero indicates an insignificant element, while a number close to one tells us that we definitely want to preserve it. In the above formula W_f indicates the weight matrix and b_f the bias value. These parameters are learned by the network during the training phase.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{6}$$

this gate is called the **input gate layer**, which decides which values we are going to update in the cell state.

$$\widetilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{7}$$

after deciding which values to update, we need to generate new candidate values to add to the state.

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t \tag{8}$$

here we update the old cell state C_{t-1}, thus producing the new cell state C_t.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{9}$$

$$h_t = o_t * tanh(C_t) \tag{10}$$

the last step is to create the actual output of the cell. Through o_t we decide which values of the state cell will be output. Then the cell state values are pushed between `-1` and `1` through *tanh* function and by means of the element-wise multiplication $*$ we output only those values we decided with o_t.

The memory capability of the LSTM-based RNNs acquires a very significant importance when it comes to written text, which is our case of study, since the ability to correctly interpreting one word is strongly based on the knowledge of the context, i.e. the rest of the sentence. A further extension of this feature is represented by the choice of making the LSTM layer **bidirectional**. The idea is pretty simple: the recurrent layer is duplicated, resulting in two layers put "side by side". The sequential input is then provided both in its original form and in a **reversed** one. This is based on the idea that, in addition to the knowledge of the past, also the knowledge of the **future** can be exploited for correctly interpreting the information that is currently dealt with.

4.2 Convolutional Neural Networks

As with the previous model, CNNs have been used to solve various types of problems. One of the first applications of this neural-architecture was in the visual domain, where they were employed for image and video recognition. However, their versatility has led them to be used in Recommendation Systems [27] and even in Natural Language Processing tasks [7]. In contrast to classic Multilayer Perceptron Neural Networks, CNNs have distinguished themselves through three essential characteristics, namely:

- **number of parameters to train**: in classic neural network architectures, i.e. those with a lot of dense hidden layers and units, the number of parameters to be trained grows really fast. With CNNs instead, the amount of train parameters is given by the size and quantity of convolution filters, which in some cases are drastically less
- **parameters sharing**: a feature detector that's useful in one part of the sentence is probably useful in another part of the sentence
- **sparsity of connections**: in each convolution layer, each output value is depends only on a small number of inputs

Those type of networks are based on the idea that the tweet text input, which is put in form of a matrix of word embeddings, can be analyzed by means of a **sliding window** of a fixed size, which runs step by step through the matrix, overlapping on different word embeddings, thus on different words of the original text. For each step, a new single feature is computed by a linear combination of the elements covered by the window and the weights of the corresponding filter. Usually, after the convolution, a **pooling** layer is applied, which in our case consists of a Max Pooling, whose objective is to preserve only the most important features by taking the maximum from a well-defined set of features contained in the previous layer. The objective of the network is to learn the weights of the various filters that have been used.

4.3 Softmax, Loss Function and Regularization

An additional fully connected softmax layer is added to the output layer of each of the two models presented in the previous Subsections. It is responsible for computing the probability distribution over the classes(emojis):

$$p(y = j|\mathbf{x}) = \frac{e^{\mathbf{x}^T \theta_j}}{\sum_{n=1}^{N} e^{\mathbf{x}^T \theta_n}} \tag{11}$$

where j is the target label and θ_n is the weight vector of the n-th class. The vector $\mathbf{x}$ instead, represents the abstract representation of the input tweet text before the last softmax layer. Both models have been trained to minimize the cross-entropy cost function shown in the following Equation:

$$Cost = -\log \prod_{i=1}^{M} p(y_i|\mathbf{s}_i) = -\sum_{i=1}^{M} [y_i \log \mathbf{s}_i + (1 - y_i) \log(1 - \mathbf{s}_i)] \tag{12}$$

where $\mathbf{s}$ is the output of the softmax layer and θ contains all the parameters optimized by the network.

To avoid having a high variance, that is, a model that would learn a decision function so complex that it could overfit the training, we decided to increase the cost function with **L2-norm** regularization terms for the parameters of the network. The new cost function is updated as follows:

$$Cost = -\log \prod_{i=1}^{M} p(y_i|\mathbf{s}_i) + \lambda||\theta||_2^2 = -\sum_{i=1}^{M}[y_i \log \mathbf{s}_i + (1-y_i)\log(1-\mathbf{s}_i)] + \lambda||\theta||_2^2 \tag{13}$$

where θ contains all the parameters optimized by the network.

As a further method of regularization we decided to use the **Dropout** technique, which prevents feature co-adaptation by setting to zero (dropping out) a portion of hidden units during the forward propagation phase [26]. In order to be able to decide where to stop training the models, we have adopted an **Early Stopping** strategy. This technique can be seen as an additional method of regularization that prevents overfitting. We extracted a **validation set** from the training set and monitored its improvements in terms of F_1-score. We then set a **patience** value of `2`, which means that if there is no improvement for three epochs in a row, the training ends.

5 Experiments

We performed an extensive set of **experiments**, intended to tune our models and identify the combination of hyper-parameters and word embeddings which would lead to the best results.

5.1 Word embeddings

The first important aspect to take into account when applying a Neural Network model to textual data is is the logic according to which words are **embedded** for the system. For our set of experiments, we have decided to apply three different strategies:

- **randomly initialized word embeddings**: the embeddings for the word are initialized randomly, and subsequently learned by the network together with the weights
- **GloVe**[11] **embeddings**: the embeddings are those provided by the GloVe algorithm[22], and they are maintained over the whole process
- **trainable GloVe embeddings**: the embeddings are those provided by the GloVe algorithm, but they can then be modified by the learning process of the network

[11] **GloVe**: `https://nlp.stanford.edu/projects/glove/`

5.2 Neural Networks setup

In this subsection, we will describe the combination of Neural Network parameters which eventually worked best, after an iterative experimental process performed with respect to the aspects described in Section 4. The models are fit on the data, including as parameter the weights of the classes, computed according to their occurrences in the dataset. The training is performed over a maximum of `50` epochs, with batches of size `2048`.

Layer	Parameters	Value
Embedding	`Input dimension`	`301784`
	`Output dimension`	`200`
	`Weights`	`embedding matrix`
	`Maximum sequence length`	`40`
	`Trainable`	`True`
	`Embeddings regularizer`	`l2 regularizer (0.000001)`
Dropout	`Percentage`	`40%`
1D convolution	`Filters`	`512`
	`Kernel size`	`5`
	`Activation`	`relu`
	`Padding`	`same`
	`Kernel regularizer`	`l2 regularizer (0.00001)`
1D max pooling	`Pool size`	`5`
Flatten	/	/
Dropout	`Percentage`	`40%`
Dense	`Number of classes`	`20`
	`Activation`	`softmax`

Table 2: CNN model

Layer	Parameters	Value
Embedding	`Input dimension`	`301784`
	`Output dimension`	`200`
	`Weights`	`embedding matrix`
	`Maximum sequence length`	`40`
	`Trainable`	`True`
	`Embeddings regularizer`	`l2 regularizer (0.000001)`
Dropout	`Percentage`	`40%`
Bi-LSTM	`Output space dimensionality`	`64`
Dense	`Number of classes`	`20`
	`Activation`	`softmax`

Table 3: Bi-LSTM model

As for the CNN scheme, we came up with a model that has a 1D convolution layer at its core, and it is structured as shown in Table 2. The RNN has a Bi-LSTM layer as its core, and it is structured as shown in Table 3. Both models are then compiled using **cross entropy** as loss function, **accuracy** as metric and `Adam` as optimizer, with the following parameters:

– learning rate: 0.001
– β_1: 0.9
– β_2: 0.999
– learning rate decay over each update: 0.001

Experiment	Accuracy	Precision	Recall	F1
SVM Baseline	0.4264	0.3235	0.2936	0.2996
CNN (no GloVe)	0.4396	0.3533	0.2982	0.2913
CNN (fixed GloVe)	0.4123	0.2959	0.2818	0.2660
CNN (trainable GloVe)	0.4549	0.3553	0.3264	0.3248
LSTM (no GloVe)	0.4391	0.3156	0.2970	0.2881
LSTM (fixed GloVe)	0.4143	0.3146	0.2709	0.2630
LSTM (trainable GloVe)	0.4357	0.3562	0.3105	0.3124
#1 Tübingen-Oslo [8]	0.4709	0.3655	0.3622	0.3599
#2 NTUA-SLP [4]	0.4474	0.3453	0.3800	0.3536
#3 hgsgnlp [1]	0.4555	0.3500	0.3357	0.3402
#4 EmoNLP [19]	0.4746	0.3943	0.3370	0.3367
#5 ECNU [20]	0.4630	0.3517	0.3311	0.3335
#6 UMDuluth-CS8761 [5]	0.4573	0.3980	0.3137	0.3183
#7 SemEval Baseline [1]	0.4256	0.3034	0.3300	0.3098

Table 4: Table of results

6 Results and discussion

In this section, we will be describing and discussing the results we have obtained by applying the models described in the previous sections. For each subtask we have performed a complete set of experiments, tuning the different parameters involved in the algorithm. Here we will report the final results for the baselines and each embedding strategy adopted for the Neural Network approach. We also include in the table the scores achieved by the top 5 participants of the SemEval competition, for a more interesting comparison. The results are show in Table 4. As we can see, the `fastText` implementation outperformed the SVM one, providing a solid baseline to start our work from. The Deep Learning approach has generally outperformed both the baselines, indicating the trainable GloVe embeddings as the best choice. Between the two Neural Network strategies, the CNN model turned out to work better than the Bi-LSTM one. As for the comparison with the SemEval ranking, as we can see, our algorithm would earn the 6th position.

7 Conclusions

The task described in this paper consisted in studying the relation between written text and **emojis**, with particular respect to the scope of the social network Twitter. This was done by implementing and evaluating a Machine Learning based tool capable of predicting, given the text of a tweet, its most likely related emoji, according to the task proposed for the SemEval 2018 competition.

In order to achieve this target, we first implemented a SVM-based **baseline**, comparing it with the `fastText` algorithm. Subsequently, we have implemented two different Neural Network architectures, based respectively on **CNNs** and **Bi-LSTM RNNs**. Our baseline was outperformed by the `fastText` based one, and together they provided a fair starting point. The Neural Networks themselves outperformed the baseline, indicating the CNN scheme and the GloVe word embeddings as the best choice. The results obtained turned out to be positively competitive with respect to the ranking of the competition, since they would place our work in the `6th` position, with `47` total participants. This shows us that simple deep neural networks can also reach quite good results by improving them through fine-tuning of hyper-parameters, regularization, and optimization of the models themselves. In terms of future work, it would be interesting to improve the results by more thoroughly tuning the hyper-parameters of the Neural Networks or by exploring new models, and to participate to the next edition of the SemEval competition.

References

1. Francesco Barbieri, Jose Camacho-Collados, Francesco Ronzano, Luis Espinosa Anke, Miguel Ballesteros, Valerio Basile, Viviana Patti, and Horacio Saggion. Semeval 2018 task 2: Multilingual emoji prediction. In *Proceedings of The 12th International Workshop on Semantic Evaluation*, pages 24–33. Association for Computational Linguistics, 2018.
2. Francesco Barbieri, Luís Marujo, Pradeep Karuturi, William Brendel, and Horacio Saggion. Exploring emoji usage and prediction through a temporal variation lens. *CoRR*, abs/1805.00731, 2018.
3. Angelo Basile and Kenny W. Lino. Tajjeb at semeval-2018 task 2: Traditional approaches just do the job with emoji prediction. pages 470–476, 01 2018.
4. Christos Baziotis, Athanasiou Nikolaos, Athanasia Kolovou, Georgios Paraskevopoulos, Nikolaos Ellinas, and Alexandros Potamianos. Ntua-slp at semeval-2018 task 2: Predicting emojis using rnns with context-aware attention. In *SemEval@NAACL-HLT*, 2018.
5. Jonathan Beaulieu and Dennis Asamoah Owusu. Umduluth-cs8761 at semeval-2018 task 2: Emojis: Too many choices? In *Proceedings of The 12th International Workshop on Semantic Evaluation*, pages 400–404. Association for Computational Linguistics, 2018.
6. Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.
7. Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 160–167, New York, NY, USA, 2008. ACM.
8. Çağrı Çöltekin and Taraka Rama. Tübingen-oslo at semeval-2018 task 2: Svms perform better than rnns in emoji prediction. In *Proceedings of The 12th International Workshop on Semantic Evaluation*, pages 34–38. Association for Computational Linguistics, 2018.
9. Alexis Conneau, Holger Schwenk, Loc Barrault, and Yann Lecun. Very deep convolutional networks for text classification. pages 1107–1116, 01 2017.

10. Jol Coster, Reinder Gerard Dalen, and Nathalie Adrinne Jacqueline Stierman. Hatching chick at semeval-2018 task 2: Multilingual emoji prediction. pages 445–448, 01 2018.
11. Bjarke Felbo, Alan Mislove, Anders Sgaard, Iyad Rahwan, and Sune Lehmann. Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. 08 2017.
12. Tim Highfield and Tama Leaver. Instagrammatics and digital methods: studying visual social media, from selfies and gifs to memes and emoji. *Communication Research and Practice*, 2(1):47–62, 2016.
13. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
14. J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79(8):2554–2558, April 1982.
15. Shuning Jin and Ted Pedersen. Duluth urop at semeval-2018 task 2: Multilingual emoji prediction with ensemble learning and oversampling. In *Proceedings of The 12th International Workshop on Semantic Evaluation*, pages 482–485. Association for Computational Linguistics, 2018.
16. Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.
17. Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.
18. Yann LeCun and Yoshua Bengio. The handbook of brain theory and neural networks. chapter Convolutional Networks for Images, Speech, and Time Series, pages 255–258. MIT Press, Cambridge, MA, USA, 1998.
19. Man Liu. Emonlp at semeval-2018 task 2: English emoji prediction with gradient boosting regression tree method and bidirectional lstm. pages 390–394, 01 2018.
20. Xingwu Lu, Xin Mao, Man Lan, and Yuanbin Wu. Ecnu at semeval-2018 task 2: Leverage traditional nlp features and neural networks methods to address twitter emoji prediction task. In *Proceedings of The 12th International Workshop on Semantic Evaluation*, pages 433–437. Association for Computational Linguistics, 2018.
21. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.
22. Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.
23. Horacio Saggion, Miguel Ballesteros, and Francesco Barbieri. Are emojis predictable? In *EACL*, 2017.
24. M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *Trans. Sig. Proc.*, 45(11):2673–2681, November 1997.
25. Aliaksei Severyn, Massimo Nicosia, Gianni Barlacchi, and Alessandro Moschitti. Distributional neural networks for automatic resolution of crossword puzzles. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 199–204, 2015.
26. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

27. Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2643–2651. Curran Associates, Inc., 2013.
28. Wieslaw Wolny. Emotion analysis of twitter data that use emoticons and emoji ideograms. In *ISD*, 2016.
29. Luda Zhao and Connie Zeng. Using neural networks to predict emoji usage from twitter data.