



Nasekin, S. and Chen, C. Y.-H. (2020) Deep learning-based cryptocurrency sentiment construction. *Digital Finance*, 2, pp. 39-67. (doi: [10.1007/s42521-020-00018-y](https://doi.org/10.1007/s42521-020-00018-y)).

This is the author's final accepted version.

There may be differences between this version and the published version. You are advised to consult the publisher's version if you wish to cite from it.

<http://eprints.gla.ac.uk/214034/>

Deposited on: 17 April 2020

Enlighten – Research publications by members of the University of Glasgow
<http://eprints.gla.ac.uk>

Deep learning-based cryptocurrency sentiment construction

Sergey Nasekin, ^{*} Cathy Yi-Hsuan Chen, [†]

January 29, 2020

Abstract

We study investor sentiment on a non-classical asset such as cryptocurrency using machine learning methods. We account for context-specific information and word similarity by using efficient language modelling tools such as construction of featurized word representations (embeddings) and recursive neural networks (RNNs). We apply these tools for sentence-level sentiment classification and sentiment index construction. This analysis is performed on a novel dataset of 1220K messages related to 425 cryptocurrencies posted on a microblogging platform StockTwits during the period between March 2013 and May 2018. Both in- and out-of-sample predictive regressions are run to test significance of the constructed sentiment index variables. We find that the constructed sentiment indices are informative regarding returns' and volatility predictability of the cryptocurrency market index.

Keywords: sentiment analysis, lexicon, social media, word embedding, deep learning, RNN

JEL Classification: G41, G4, G12

1 Introduction

Classical asset pricing theories, mainly relying on the concept of arbitrage, face challenges in the face of a surge of new asset classes like cryptocurrencies. Compared to classical financial assets whose fundamental value can be determined from cash flows such as dividends and earnings, fundamental value of cryptocurrencies is harder to grasp. The techniques behind cryptocurrencies, such as blockchain, ICOs (Initial Coin Offerings), decentralized schemes, complicate fair price estimation.

Sentiment plays a significant role in price evolution, given possible arbitrage opportunities and "intangible" fundamental values, see Aboody et al. (2018). Therefore reliable measurement of

^{*}Deutsche Bank AG, (*e-mail: sergey.nasekin@gmail.com*)

[†]Adam Smith Business School, University of Glasgow(*e-mail: CathyYi-Hsuan.Chen@glasgow.ac.uk*)

cryptocurrency sentiment is particularly of interest. Sentiment provides explanatory power on firms' future performance, especially when fundamental information is incomplete or biased, see Tetlock et al. (2008); Lerman and Livnat (2010); Feldman et al. (2010); Loughran and McDonald (2011).

There has been a growing body of research recently on predictability of markets' dynamics using information distilled from textual data sources. These include studies on traditional asset markets' prediction, see Yu (2013), Plakandaras et al. (2015), Nassirtoussi et al. (2015), Persio and Honchar (2016) as well as cryptocurrencies, see Mai et al. (2018), Cheuque Cerda and L. Reutter (2019).

News about cryptocurrencies, similar to financial news about stock markets, can be used to construct sentiment. We use a body of text messages from social media used by the crypto community to collect representative opinions on cryptocurrency trends. In particular, we use StockTwits, a leading social media platform for traders and investors. Messages on the StockTwits platform appear to have predictive value for returns; as found by Chen et al. (2018), sentiment distilled from StockTwits' messages performs better with respect to return predictability compared to sentiment obtained from discussions on Reddit focusing on crypto related topics. Microblogging users tend to react promptly to events, news and other possibly non-public information, which allows real-time sentiment assessment.

We collect large amounts of text messages from StockTwits and then use a deep learning method to distil the sentiment in the crypto community. This approach allows to appropriately account for the domain-specific terms widely used by this community, such as "mining", "blockchain", "ICO", "wallet", "shitcoin", "binance", "hodl" as well as non-text characters that convey emotions, such as emojis and emoticons.

A pioneer study by Chen et al. (2018) demonstrates the performance of cryptocurrency-specific sentiment, with a domain-specific lexicon created by the TF-IDF (Term Frequency - Inverse Document Frequency) scheme. However, the traditional bag-of-words approach does not account for context-specific dependencies between words and therefore important information about semantic structure of the sentence is lost. We therefore employ deep recurrent neural network (RNN) sequence models such as LSTM and GRU to learn long-term semantic and syntactic dependencies in the messages. We augment these models with Word2Vec-based embeddings to speed up calculation and allow for domain-based lexicon construction.

We create crypto-sentiment indices using a RNN predictions as well as utilizing an approach of lexicon expansion. We extend general social media lexica by Renault (2017) viewed as seed lexica by incorporating domain-specific terms with a certain degree of similarity in their word embeddings. The indices we create demonstrate predictive ability with respect to logarithmic returns of a cryptocurrency index "CRIX" developed by Trimborn and Härdle (2018) as well as its volatility. We expand standard predictive regression models for autoregressive mean and variance to show statistical significance of sentiment variables.

2 Language modelling framework

In this section we describe traditional and new approaches to language modelling. Several advances in natural language processing have been done recently which deliver superior performance

compared to traditional methods such as the bag-of-words approach. Most of them are based on deep learning and do not require extensive feature engineering but are able to extract valuable information via smart analysis of context dependencies, feature embeddings and autoregressive factorization of textual information.

2.1 Modern approaches to language modelling

We can differentiate between classical and more recent deep-learning-based approaches to natural language modelling for a variety of tasks. Tasks usually addressed by NLP are text classification, word sequence modelling/next word prediction, text meaning extraction, parts-of-speech tagging. These tasks are addressed by such classical models as bag-of-words representation combined with linear/nonlinear classifiers (logistic regression/SVM), naive Bayes, hidden Markov models.

More advanced tasks such as machine translation, named entity recognition require understanding of long-range dependencies between words in a context. Such state-of-the-art techniques as neural language models (recurrent RNN models, discussed in more detail below) and more recently introduced attention-based models such as BERT and XLNet. The Bidirectional Encoder Representations from Transformers (BERT) proposed recently by Devlin et al. (2018), have been shown to achieve state-of-the-art results on a wide variety of NLP tasks. They apply a masked language model in a multi-layer bidirectional "transformer" encoder proposed by Vaswani et al. (2017) to generate context-specific word and sentence representations. Working with a BERT model involves pre-training on a specified corpus and fine-tuning of all parameters for a specific task. Additional layers e.g. classification layers can be trained for the purpose of sentiment analysis. An advantage of BERT is its usage of the multi-headed attention mechanism which gathers information about the relevant context of a word and represents it in a "smart" embedding. A disadvantage is a discrepancy between pre-training and fine-tuning phases when the "masked" tokens used in training do not occur during fine-tuning.

Another disadvantage of BERT is that while the predictions for the masked tokens are made in parallel, dependencies between the predictions are not learned. The XLNet approach proposed by Yang et al. (2019) aims to improve on this aspect by capturing bidirectional context and simultaneously avoiding masking and independent parallel predictions. The XLNet model has three main elements: first, bidirectional context is captured by means of permuting the autoregressive factorization order of the input sequence. Positional encoding is used to keep track of the original sequence order. Second, the training is improved using the so-called two-stream self-attention mechanism: the usual self-attention mechanism described in Vaswani et al. (2017) is separated into content and query streams and this achieves blocking the embedding content while using the query stream to pass the positional encodings. Lastly, the segment recurrence mechanism and relative encoding scheme of a state-of-the-art AR language model, Transformer-XL by Dai et al. (2019), are integrated into XLNet architecture. Finally, both XLNet and BERT are quite large models with tens or even hundreds of millions of parameters which require significant computing capabilities to reproduce the SOTA results. GPUs/TPUs counted in numbers equal to batch size, with large amounts of RAM have been used for training and fine-tuning the pre-trained models.

Modern deep learning-based methods are known to constantly improve their performance as the amount of training data increases, unlike traditional learning approaches. This happens because the usual bias-variance trade-off becomes less of a problem as the complexity of neural models can

be set up arbitrarily large and as long as there is proper regularization in place and a large training data set is available, both bias and variance can be reduced simultaneously without hurting each other. This as well as the advantage of parallelization of mini-batch gradient descent and architecture flexibility make modern deep learning-based methods the tool of choice for language modelling when large amounts of text data can be harvested from online messaging services.

2.2 Recurrent neural network models

Recurrent neural networks (RNN) are quite effective for natural language processing tasks because they have "memory". They can read inputs such as words one at a time, and remember some information/context through the hidden layer activations that are passed from one time step to the next one. This allows a uni-directional RNN to take information from the past to process later inputs. Another type of RNN, a bi-directional RNN, can process context from both the past and the future time steps.

Basic RNNs are known to suffer from the problem of "vanishing gradients". This problem causes the simple RNN to fail at capturing long-period syntactical dependencies in sentences, for instance, between words, which are located far from each other but still having a meaningful connection. Therefore other refined versions of RNN have been proposed, such as the LSTM (long short-term memory) and GRU (gated recurrent unit) schemes, introduced by Hochreiter and Schmidhuber (1997) and Cho et al. (2014), respectively. Building blocks of these RNN types are connected by memory states which are activated or de-activated using *gates*. This modification enables GRU and LSTM networks to "keep track" of syntactical/grammatical structures and remember them for many time steps.

A general architecture of a sentiment prediction LSTM/RNN network is presented in Figure 1. It consists of the input sequence, an embedding lookup matrix, several layers of LSTM/GRU cells/units, an output sequence, mean pooling and softmax layers. The core of this structure are the LSTM or GRU cells. Structures of these cells are presented in Figures 2, 3, respectively.

The LSTM architecture in Figure 2 introduces the "cell state" c_t which is the memory of the cell: it is able to keep information about the previous states of LSTM cells. The amount of information stored in the cell state is controlled by the "gates": a forget gate f_t , an input gate i_t , and an output gate o_t . The gates, in their turn, depend on the current input (vector representation of a word) x_t and the information about the previous cell contained in the "hidden state" h_t .

At time step t , the forget gate f_t determines how much of the previous state c_{t-1} will be kept based on the values of the previous hidden state h_{t-1} and the current input x_t :

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \quad (1)$$

where the sigmoid function $\sigma(x) = 1/\{1 + \exp(-x)\}$ outputs a value between 0 and 1 for each number in the cell state c_{t-1} .

The LSTM cell generates an update to c_{t-1} through a new candidate value of the cell state, \tilde{c}_t , which also depends on x_t and h_{t-1} and is created using a tanh activation:

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c), \quad (2)$$

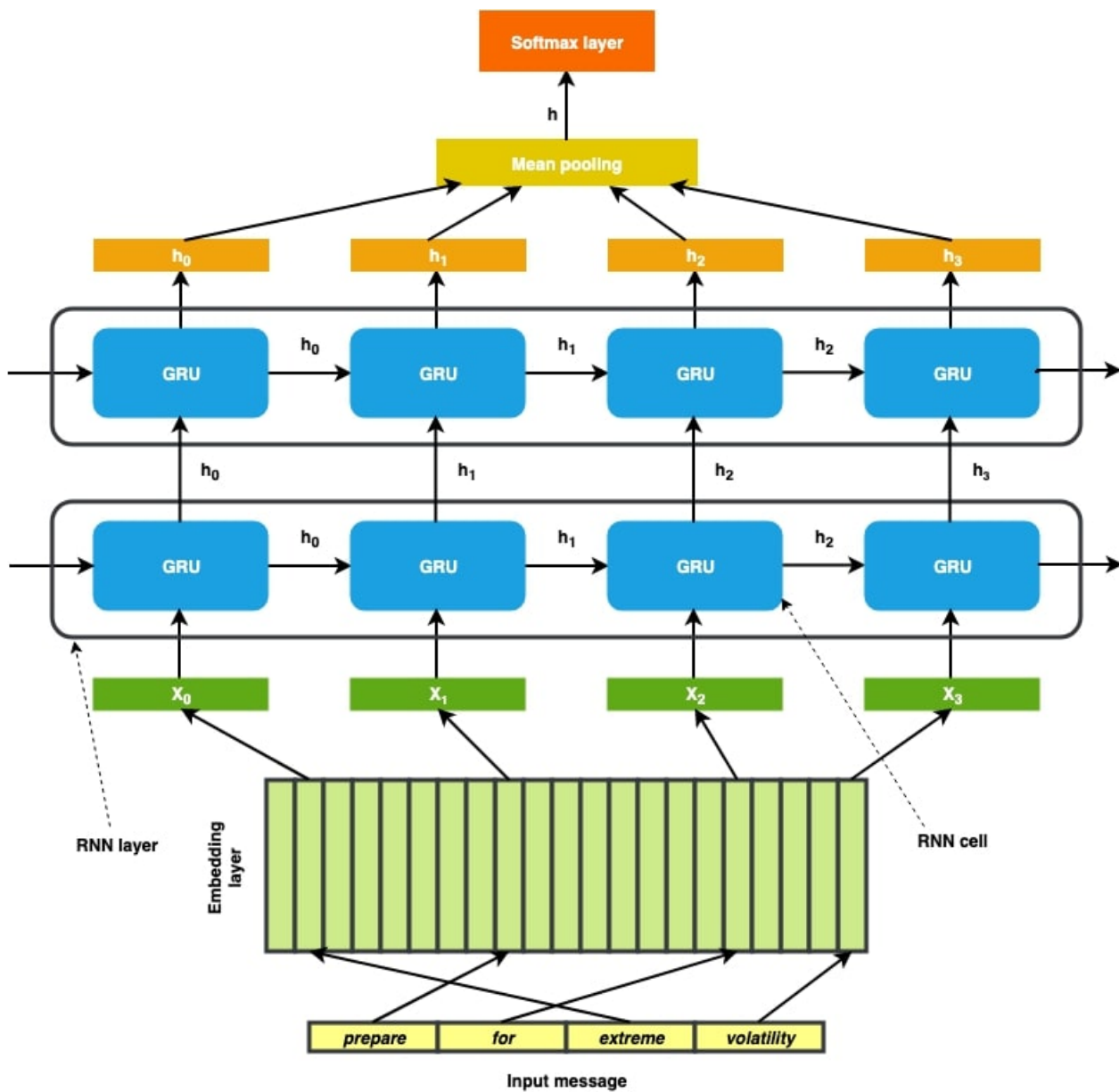


Figure 1: General architecture of an RNN

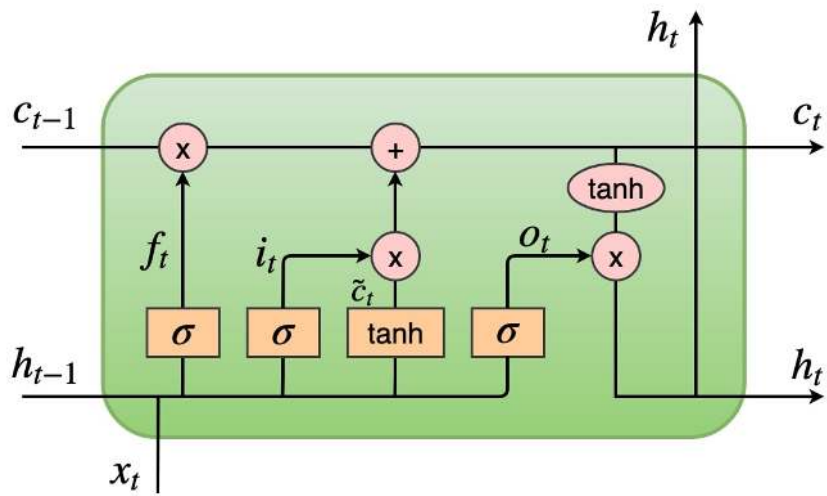


Figure 2: Structure of an LSTM unit

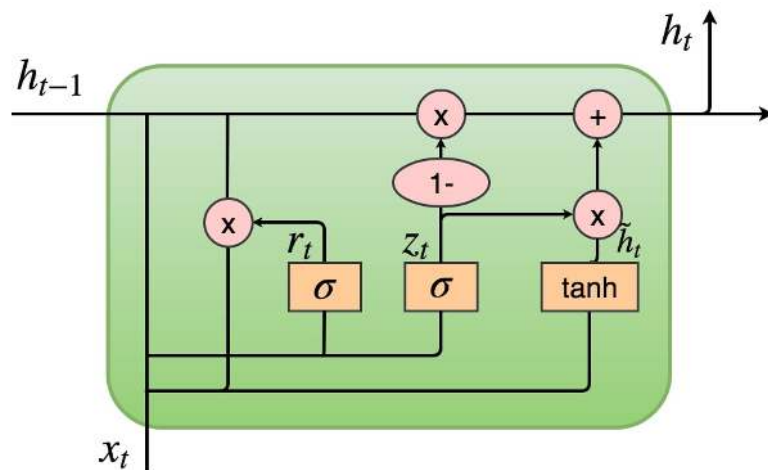


Figure 3: Structure of a GRU unit

where $\tanh(x) = \{\exp(x) - \exp(-x)\} / \{\exp(x) + \exp(-x)\}$.

To decide what will be stored in the next cell state c_t , it is then decided, "how much" of the new candidate state \tilde{c}_t will be fed into c_t . This is done through the input gate i_t , which, analogously to the situation with the forget gate f_t , outputs a number between 0 and 1 for each value of \tilde{c}_t :

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i). \quad (3)$$

An updated value of the cell state c_t is then a weighted sum of the previous cell state value c_{t-1} and the new candidate value \tilde{c}_t with f_t and i_t as weights:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \quad (4)$$

where \odot denotes element-wise multiplication.

Finally, the next value of the hidden state h_t is a "filtered" value of the cell state c_t , which is put through the tanh activation and multiplied element-wise by the values of the output gate o_t :

$$h_t = o_t \odot \tanh(c_t), \quad (5)$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o). \quad (6)$$

The resulting hidden state value h_t is propagated along LSTM cells within LSTM units as well as between the units and also upwards to the next hidden layer. The final output of a unit is a sequence h_0, h_1, \dots, h_T which is fed into the next layer of a deep LSTM network.

Deep LSTM network architecture captures long-term dependencies efficiently. It essentially balances information from older and newer contexts through employing the memory state: hence the name "long short-term memory".

A similar system of equations describes the GRU architecture, which is a more parsimonious representation of a gated RNN unit with memory. In a GRU unit shown in Figure 3, there is a "reset gate" r_t and an "update gate" z_t . The latter merges the forget and input gates of an LSTM unit together. Also the cell and hidden states are combined into one hidden state h_t .

The reset gate r_t controls the amount of past information contained in h_{t-1} which is kept at time step t . Similarly to LSTM, The update gate z_t is used to update h_t through filtering h_{t-1} and the new candidate value \tilde{h}_t . The system of equations describing the GRU architecture is as follows:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z), \quad (7)$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r), \quad (8)$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h r_t \odot h_{t-1} + b_h), \quad (9)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t. \quad (10)$$

Parameters of LSTM and GRU networks such as $W_o, W_i, W_f, W_c, U_o, U_i, U_f, U_c, b_o, b_i, b_f, b_c$ and others are found via a "backpropagation through time" (BPTT) procedure. This procedure accumulates gradients along the input sequence x . We provide the details of the BPTT procedure for the LSTM architecture in Appendix 6. Very similar results can be found by reasoning along the same lines for the simpler case of GRU.

2.3 Representation of sentence inputs

To build a language model like a sentiment prediction model for messages, input sentences must have vector representation. The simplest approach would be to use one-hot vector representation, where the dimension of representation vectors is the size of the vocabulary which may be quite large.

Besides being costly to train, the language model relying on one-hot representations of terms in the input sentence is not able to find meaningful relations between words. Therefore it makes sense to obtain vector representations of words in a space of features of lower dimensionality than the vocabulary size. Dense representations obtained in a context-oriented manner are able to better reflect meaningful word analogies between words.

Several approaches have been proposed to learn dense feature vector representations of words (word embeddings) such as latent semantic analysis, see Deerwester et al. (1990) or neural language model methods. The former relies on singular value decomposition (SVD) to extract word embeddings from high-dimensional term-document matrices. The latter tries to predict a word in a sentence using a fixed "window" of context words via a feed-forward neural network with a softmax output activation where the softmax goes over all the words in the vocabulary. The final output of such a model is not the prediction itself, but the embedding vectors which have been learned in the process.

The disadvantages of these methods are easy to see: the former approach requires significant computational cost if the term-document matrix is large; furthermore, context is not explicitly accounted for. In the latter method the primary hurdle is also the computational cost of training a high-dimensional softmax classifier. One of the most popular proposed solutions to this problem is the Word2Vec model introduced by Mikolov et al. (2013). It covers two ways of generating dense embeddings: skip-gram and CBOW (continuous bag of words). In skip-gram, one predicts the context words c_1, c_2, \dots, c_C from a given word w . In CBOW, a word w is predicted from the context c_1, c_2, \dots, c_C .

Let us consider the more popular skip-gram model to illustrate Word2Vec approach. The goal is to maximize the conditional probability $p(c|w; \theta)$ of obtaining context words given the current word; this probability can be parameterized as a softmax:

$$p(c|w; \theta) = \frac{e^{v_c \cdot v_w}}{\sum_{\substack{c' \in C \\ c' \neq c}} e^{v_{c'} \cdot v_w}}, \quad (11)$$

where v_c and $v_w \in \mathbb{R}^d$ are vector representations or *embeddings* of c and w , respectively. The parameters θ are v_{c_i}, v_{w_i} for $w \in P, c \in C$, where P and C are the vocabulary and the set of all contexts, respectively. The objective function to maximize is therefore

$$\arg \max_{\theta} \prod_{w \in P} \prod_{c \in C} p(c|w; \theta). \quad (12)$$

In fact, the objective in (12) is not practical as it is very computationally expensive to compute because of the summation over all c' in the denominator in (11). One popular solution to this problem is to use the so-called negative sampling which randomly samples several "noise" words

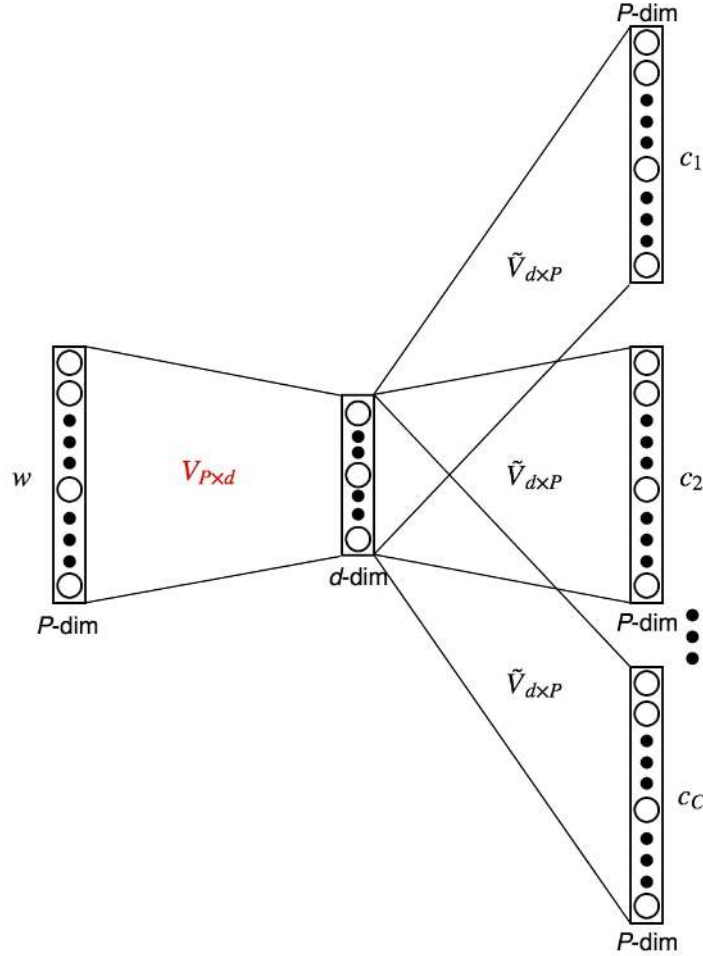


Figure 4: Structure of a Word2Vec neural model

from the corpus based on their frequency. This amounts to generating "normal" and "noise" pairs $(w, c) \in D$ and $(w, c) \in D'$, respectively, where $D \cup D'$ comprises the entire corpus. Then the negative sampling objective can be formulated as follows:

$$\arg \max_{\theta} \sum_{(w,c) \in D} \log \sigma(v_c \cdot v_w) + \sum_{(w,c) \in D'} \log \sigma(-v_c \cdot v_w), \quad (13)$$

where σ is the softmax function. For more details, see Goldberg and Levy (2014).

The Word2Vec model is graphically represented as a shallow neural network model with one hidden layer with shared weights \tilde{V} for all context words c ; see Figure 4. In fact, the objective in (13) is an approximation to the original objective (12) and as such does not give optimal predictions for context words, but tends to produce meaningful embeddings V which can be further used in training a deep RNN model. The RNN architecture shown in Figure 1 has an embedding layer which can be learned by the RNN training procedure. On the other hand, this layer can be separately pre-trained via the Word2Vec algorithm and then inserted into the RNN and used either as fixed parameters or an initialization method. Thus reduction in computational cost can be achieved or the RNN training algorithm will converge more quickly if the embedding layer is pre-trained.

3 Data and preprocessing

3.1 StockTwits

StockTwits¹ is a microblogging platform popular among investors and traders. It bears resemblance to Twitter, but is more focused on finance-related topics. One of the features contributing to its popularity is availability of messages and user streams related to financial assets (including cryptocurrencies), generated by investors. According to StockTwits, more than one million users now use the platform to share information and ideas, reaching an audience of more than 40 million people across the financial web and social media. Conversations are organized around "cashtags" which combine the dollar sign and tickers into unique identifiers for financial assets (e.g. \$SPY for S&P 500) and allow indexing people's thoughts and ideas about specific assets. Users can also express their sentiment by labeling messages as "Bearish" (negative) or "Bullish" (positive) via a toggle button. Such labeled data benefits offer a convenient data source for textual analysis in the context of supervised learning.

Since 2014 StockTwits have added streams and symbology for cryptocurrencies and tokens, from 100+ in the beginning to 400+ recently. This new vibrant asset class has successfully attracted huge attention from a large investor community and newcomers. New cryptocurrencies are regularly added to the list of cashtags supported by StockTwits.² A cashtag refers to a cryptocurrency if it ends with ".X" (e.g. \$BTC.X for Bitcoin, \$LTC.X for Litecoin).

It must be noted that Twitter in 2012 also launched cashtags allowing its users to follow stocks and later cryptocurrencies. StockTwits' cashtag feature however has existed longer as part of their interface and has more functionality. It is also much more user-friendly for the finance-focused community. For instance, StockTwits not only give the related cashtag streams, but also provide assets' current trading information, as well as a visual representation of assets' prices over time via an interactive graph. On StockTwits, there is an option of sorting for charts and videos. Most importantly, on Twitter there is no option of sentiment labeling by the user which is crucial for obtaining a dataset suitable for supervised learning purposes. All this and some other useful features demonstrate a pronounced focus on finance experts which is more likely to attract a larger and more relevant community of traders and investors who can provide informed opinions on the asset price direction. For these obvious reasons we choose to use StockTwits and not Twitter as the data source.

We use StockTwits' application programming interface (API) to download all messages containing a cashtag referring to a cryptocurrency. StockTwits API also provides the user's unique identifier for each message, the time it was posted with one-second precision, and the sentiment associated by the user ("Bullish", "Bearish" or unclassified). Our final dataset contains 1,220,728 messages from 33,613 distinct users, posted between March 2013 and May 2018, and related to 425 cryptocurrencies. Overall, 472,255 messages are classified as bullish (38.6%) and 92,033 as bearish (7.5%), and the rest is unclassified. The imbalance between the numbers of positive and negative messages shows that crypto investors are optimistic on average, as previously found by Kim and Kim (2014) or Avery et al. (2016).

¹<https://stocktwits.com/>

²This list can be found at <https://api.stocktwits.com/symbol-sync/symbols.csv>.

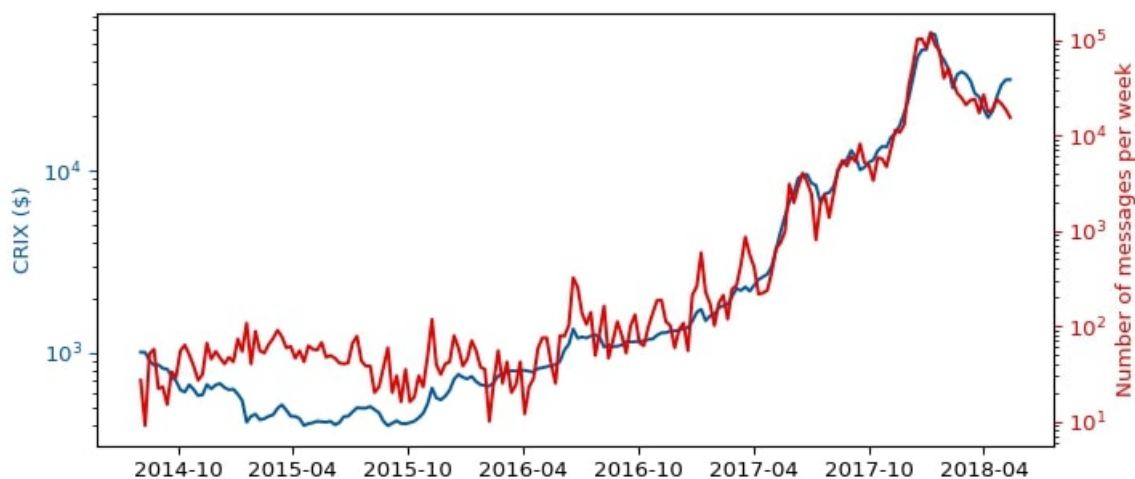


Figure 5: Weekly number of crypto-related messages on StockTwits and CRIX value (log scale).

Before processing	After processing
\$BTC.X why can't it hold \$14k ?? Shameless pumpers said 25 by Christmas 🤔	cashtag why cant it hold moneytag ? ? shameless pumpers said numbertag by christmas 🤔
\$BTC.X Merry Xmas to all coiners and no coiners alike! 2018 is gonna be lit!! 💰🎄🚀	cashtag merry xmas to all coiners and negtag_coiners alike ! numbertag is gonna be lit !! 💰🎄🚀
\$XVG.X all greeeeeeeeeb 😂😂	cashtag all greeeb 😂😂
\$NEO.X In NEO I trust!!! https://neousd.bid/	cashtag in neo i trust !!! linktag

Table 1: Pre-processing of StockTwits messages

Figure 5 shows the number of messages per week related to cryptocurrencies on StockTwits, and the weekly average of the cryptocurrency index CRIX, see Trimborn and Härdle (2018). Investor attention skyrocketed similar to the prices during the 2017 market boom, but declined as the prices dropped in 2018. This indicates a certain relationship between investors' discussion intensity on StockTwits and crypto price movement.

3.2 Preprocessing

We follow the textual data pre-processing procedure suggested by Oliveira et al. (2016) and Chen et al. (2018). First, all messages are lower-cased. To remove letter repetitions, which has been shown to be a widespread feature of sentiment expression on microblogs (Brody and Diakopoulos; 2011), sequences of repeated letters are reduced to a maximum length of 3 (for example, "Cooooool" is truncated as "cool"). Tickers (such as "\$BTC.X", "\$ETH.X", etc.), currency such as dollar or euro values, hyperlinks, numbers and mentions of users are respectively replaced by the words "cashtag", "moneytag", "linktag", "numbertag" and "usertag". Numbers are substituted

by a single tag since the whole set of distinct numbers is too large. For privacy reasons, all users and URL addresses are replaced by "usertag" and "linktag", respectively. We also exclude messages composed only of cashtags, URL links or pure punctuation. The prefix "negtag_" is added to any word consecutive to "not", "no", "none", "neither", "never" or "nobody". Stopwords with less potential function in the sentence, such as articles, prepositions, pronouns, are removed. Other stopwords such as conjunctions, common adverbs and adjectives, auxiliary verbs are retained, as they are important for the sentence structure which is learned by an RNN network. All punctuation except the characters "?" and "!" is removed. Exclamation and interrogation marks are kept as it was previously shown that they often improve lexicon accuracy, see Renault (2017) and Table 3 of Chen et al. (2018). Table 1 shows examples of messages before and after processing.

4 Language modelling and sentiment prediction

In this section, we proceed to message modelling with RNN. We define and train several RNN setups and select the best-performing one which is then used to build the sentiment index.

4.1 Supervised RNN setup

As noticed before, two features of StockTwits' messages make them useful for language modelling' research. First, they contain explicit references to the asset they are concerned with via "cashtags". This allows us to select messages which are only related to cryptocurrencies. Second, users are encouraged to report their sentiment corresponding to the posted message as "Bullish" (positive) or "Bearish" (negative), which results in a large dataset suitable for supervised learning purposes.

To avoid domination of the corpus by excessively prolific users (possibly robots), we impose a maximum proportion of 1% of the dataset per user, as suggested by Pang et al. (2002). The full filtered dataset of 528,443 messages is randomly split into a training subset of 422,755 messages (80% of all samples) as well as validation and hold-out test subsets of 52,844 messages in each one (10% of all samples in each). We use the technique of stratified sampling to ensure equal proportions of positive and negative messages in train, validation and test datasets.

Input data have an unbalanced structure: just about 16% of all labeled messages are bearish while the rest are bullish. Various methods have been proposed to address this problem:

- down-sampling the majority class,
- over-sampling the minority class,
- more advanced techniques such as SMOTE (Synthetic Minority Oversampling Technique) see Chawla et al. (2011).

We apply oversampling to under-represented bearish messages in the train subset. Oversampling is not applied to the validation and test subsets; the hold-out test subset should retain the

distribution of the real-world data, while the validation subset should have a distribution matching that of the test subset in order to optimize with respect to the test subset.

We set up recurrent neural network models using methodology from Section 2, presented in Table 2. Two different approaches are tested: in the first one embeddings are initialized randomly from the uniform distribution on the interval $[-0.1, 0.1]$. In the other they are pre-trained using the Word2Vec model of the dimension $d = 256$. The context window size is chosen to be 10; sentences of size longer or shorter than 50 words are cropped or padded with zeros to reach the length of 50.

We fix setups for LSTM and GRU RNN architectures shown in 2 which have been found to work well after trying various other hyperparameter setups. Dropout is a state-of-the-art regularization technique used to prevent overfitting neural networks of different architectures. The usual dropout approach for RNNs is to randomly switch off connections between the hidden and the output layers. Recently a recurrent dropout version has been efficiently used which drops connections between recurrent units of the network, see Gal and Ghahramani (2015).

Word2Vec			
Algorithm		Skip-gram	
Embedding dimension		256	
Context window size		10	
Number of epochs		25	
Common parameters			
Maximum encoded message length	50		
Unknown embedding vector		$U[-0.1, 0.1]$	
Loss function		Binary cross-entropy	
Batch size	32		
Number of epochs	20		
LSTM		GRU	
Recurrent layers	2	Recurrent layers	3
Recurrent unit	64	Recurrent unit	128
Recurrent dropout	50%	Recurrent dropout	50%
Dropout	50%	Dropout	50%
Activation	tanh	Activation	tanh
Optimizer	Adadelta	Optimizer	Adadelta

Table 2: Parameters' setup for RNN

4.2 Estimation results

Performance metrics of the tested RNN setups are shown in Table 3. It follows that the LSTM

		Precision	Recall	F1-score	Data
LSTM (pre-trained embeddings)	Bullish	0.94	0.87	0.90	44,233
	Bearish	0.50	0.69	0.58	8,611
	Weighted avg. / Total	0.87	0.84	0.85	52,844
GRU (pre-trained embeddings)	Bullish	0.96	0.66	0.78	44,233
	Bearish	0.32	0.85	0.47	8,611
	Weighted avg. / Total	0.86	0.69	0.73	52,844
LSTM (random embeddings)	Bullish	0.92	0.89	0.90	44,233
	Bearish	0.50	0.58	0.54	8,611
	Weighted avg. / Total	0.85	0.84	0.84	52,844
GRU (random embeddings)	Bullish	0.93	0.78	0.85	44,233
	Bearish	0.38	0.69	0.49	8,611
	Weighted avg. / Total	0.84	0.77	0.79	52,844

Table 3: Performance metrics for RNN models

setup with word embeddings pre-trained by Word2Vec demonstrates better performance than other setups. Performance is measured in terms of precision and recall which are standard measures of exactness and completeness of pattern recognition and classification algorithms. These measures are suitable for ranking classification algorithms' performance on imbalanced datasets which is also the case for the current study. Precision and recall over both classes are calculated as weighted averages where weights are percentages of bullish and bearish messages.

Assuming that a "true positive" (tp_i^{Be}) is an instance of correct prediction of a bearish message and a "true negative" is an instance of correct prediction of a bullish message, then there is a "false negative" (fn_j^{Be}) when a bullish message is classified as bearish. Accordingly, there is a "false positive" (fp_k^{Be}) when a bearish message is classified as bullish. Then precision P^{Be} and recall R^{Be} for bearish messages are defined as follows:

$$P^{Be} = \frac{\sum_i tp_i^{Be}}{\sum_i tp_i^{Be} + \sum_k fp_k^{Be}}, \quad (14)$$

$$R^{Be} = \frac{\sum_i tp_i^{Be}}{\sum_i tp_i^{Be} + \sum_j fn_j^{Be}}. \quad (15)$$

Similarly, the values of precision and recall for bullish messages can be found by "switching" the confusion categories for bearish and bullish classes.

Lower precision for bearish messages is caused by a higher rate of false positives when regarding bearish messages as positives as there are many more bullish messages ("negatives") which become

false positives.

Considering the final task of constructing the sentiment index, we prefer a lower false negatives' rate for bearish messages to avoid underestimation of risk. In other words, we would like the measure of recall to be higher for bearish messages. Intuitively, higher recall implies ability to correctly identify all the relevant samples in the dataset.

LSTM model with pre-trained and randomly initialized embeddings provide similar weighted average recall. Simultaneously the precision of the pre-trained case is higher than that of the random case; that is, the pre-trained setup predicts bearish messages with more certainty. LSTM specifications demonstrate better performance than GRU ones. Therefore the LSTM model with pre-trained embeddings is chosen for sentiment index construction.

4.3 Sentiment index construction

We create an aggregate cryptocurrency sentiment index which is designed as representative opinion from the crypto community using the StockTwits platform. We then test whether the constructed index is relevant for future market performance and can be used to predict price and volatility evolution of the cryptocurrency index CRIX.

There are several ways available to build a sentiment index in our framework. One is to use a "seed" lexicon of terms with known sentiment scores and use cosine similarity to extend this lexicon using the embedding vectors of new context-specific terms. To further illustrate this approach, we trained the Word2Vec algorithm according to the setup given in Table 2 using 528,443 messages with a vocabulary size of 9200. Examples of Word2Vec-based seed and other corpus term similarities are presented in Tables 4 and 5. We show 10 terms from both bullish and bearish seed lexica most frequently used in the corpus; for each of these terms, we show 5 most similar non-seed context-specific corpus terms according to their cosine similarities.

We use a seed dictionary of 1311 terms from Renault (2017) constructed for StockTwits forums. The terms collected in this seed lexicon are commonly used across different assets discussed on the forums and can be viewed as "finance domain general" lexicon. Each word in the lexicon has a sentiment score in the interval of $[-1, 1]$. Then we augment it with new context-specific terms as follows:

1. for each seed word d_S , find 2 most similar non-seed words d_{NS} using the trained embeddings, determined by non-negative cosine similarity value $CS(d_S, d_{NS})$ with $CS(a, b)$ defined as

$$CS(a, b) = \frac{\sum_{i=1}^L a_i b_i}{\sqrt{\sum_{i=1}^L a_i^2} \sqrt{\sum_{i=1}^L b_i^2}}, \quad (16)$$

where a, b are numeric vectors of dimension L ,

2. assign a sentiment score $SW(d_{NS})$ to the non-seed word d calculated as

$$SW(d_{NS}) = CS(d_S, d_{NS}) \times SW(d_S), \quad (17)$$

with $SW(d_S)$ actually known from the seed lexicon.

After this procedure, the seed lexicon is augmented by further 1,286 terms counting 2,597 words in total. An illustration of these additional terms is presented in Figure 6, where context-specific bullish and bearish terms are grouped into "word clouds" reflecting term frequency. The larger the font of the term in the "word cloud", the higher is the frequency. Context-specific terms emerge in the visualization and sometimes otherwise normally positive or neutral words acquire negative connotation and vice versa.

The crypto sentiment index is quantified by averaging the sentiment scores across cryptocurrency-related messages (where the cashtag ends with ".X" to indicate a crypto asset). The sentiment score of an individual message m , $score_{S/NS;m}$ is calculated by averaging the sentiment weights of both seed and non-seed cryptocurrency-specific terms within the message:

$$score_{S/NS;m} = N_m^{-1} \sum_{i=1}^{N_m} SW(d_{S/NS,m}^{(i)}), \quad (18)$$

where N_m stands for the total number of words in the message m and $d_{S/NS,m}^{(i)}$ denotes the i th d_S or d_{NS} in the message m , SW is as defined in (17). Then daily sentiment index, using equal weights of scores across messages within the same trading day t , is generated as follows:

$$sent_t^{expand} = M_t^{-1} \sum_{m=1}^{m=M_t} score_{S/NS;m} \quad (19)$$

where M_t stands for total number of messages posted during the day t . The index is smoothed using the 1-week moving average to level out idiosyncratic jumps in individual investors' opinions.












		Terms from seed dictionary									
		"buy"	"dip"	"hold"	"good"	"bears"	"long"	"higher"	"holding"	"cheap"	"love"
Most similar terms from corpus	"buy buy"	"buy dip"	"buy hold"	"all good"	"poor bears"	"going long"	"much higher"	"still holding"	"coins cheap"		
	(0.432)	(0.319)	(0.348)	(0.253)	(0.303)	(0.248)	(0.327)	(0.348)	(0.368)	(0.218)	
	"strong buy"	"on dip"	"hold hold"	"oh man"	"bulls"	"hodl long"	"symmetrical triangle"	"commission"	"so cheap"	"buy volume"	
	(0.258)	(0.301)	(0.249)	(0.244)	(0.265)	(0.232)	(0.221)	(0.206)	(0.278)	(0.211)	
	"view today"	"easy work"	"usd10380"	"really good"	"hibernating"	"lasts"	 	"sale limited"	"holly satoshi"		
	(0.246)	(0.256)	(0.230)	(0.234)	(0.260)	(0.218)	(0.215)	(0.202)	(0.211)	(0.208)	
"marketoutlook"	"more dip"	"guys hold"	"Ð"	"jealousy"	"hold long"	 	"70k"	"dumb bears"	 		
(0.228)	(0.239)	(0.219)	(0.225)	(0.243)	(0.203)	(0.215)	(0.192)	(0.204)	(0.200)		
"holding ico"	"big dip"	"hodl"	"loading zone"	"go home"		"nice consolidation"	"getting close"	"nakamoto 			
(0.228)	(0.226)	(0.216)	(0.208)	(0.233)	(0.199)	(0.211)	(0.183)	(0.198)	(0.199)		

Table 4: Most similar corpus terms related to bullish seed terms (cosine similarity in brackets)

		Terms from seed dictionary									
		"sell"	"bulls"	"dump"	"shit"	"sold"	"short"	"drop"	"crash"	"scam"	"bearish"
Most similar terms from corpus	"sell sell"	"bears"	"pump dump"	"piece shit"	"panic sold"	"big short"	"big drop"	"bitcoin crash"	"irrational"	"still bearish"	
	(0.325)	(0.265)	(0.374)	(0.292)	(0.277)	(0.261)	(0.287)	(0.261)	(0.236)	(0.293)	
	"dump dump"	"most bears"	"dump dump"	"past hours"	"glad sold"	"futures"	"go down"	"recover"	"trash"	"short term"	
	(0.243)	(0.241)	(0.280)	(0.210)	(0.248)	(0.237)	(0.224)	(0.246)	(0.232)	(0.241)	
	"🐘 🐘"	"reversal"	"ÐÐ"	"absurd"	"shoulda"	"squeezed"	"didn sell"	"flash crash"	"crooks"	"👤"	
	(0.231)	(0.239)	(0.231)	(0.209)	(0.245)	(0.215)	(0.212)	(0.217)	(0.220)	(0.226)	
	"🏠 🏠"	"📌"	"another pump"	"😂 💰"	"bought 19k"	"reversing"	"bounce"	"plummet"	"shitcoin"	"lower low"	
(0.221)	(0.222)	(0.210)	(0.200)	(0.241)	(0.210)	(0.209)	(0.209)	(0.217)	(0.216)		
"want sell"	"ha ha"	"whales"	"lolll"	"dumped"	"hey bears"	"tried warn"	"real value"	"ponzi scheme"	"red day"		
(0.220)	(0.216)	(0.198)	(0.199)	(0.221)	(0.207)	(0.207)	(0.197)	(0.213)	(0.215)		

Table 5: Most similar corpus terms related to bearish seed terms (cosine similarity in brackets)

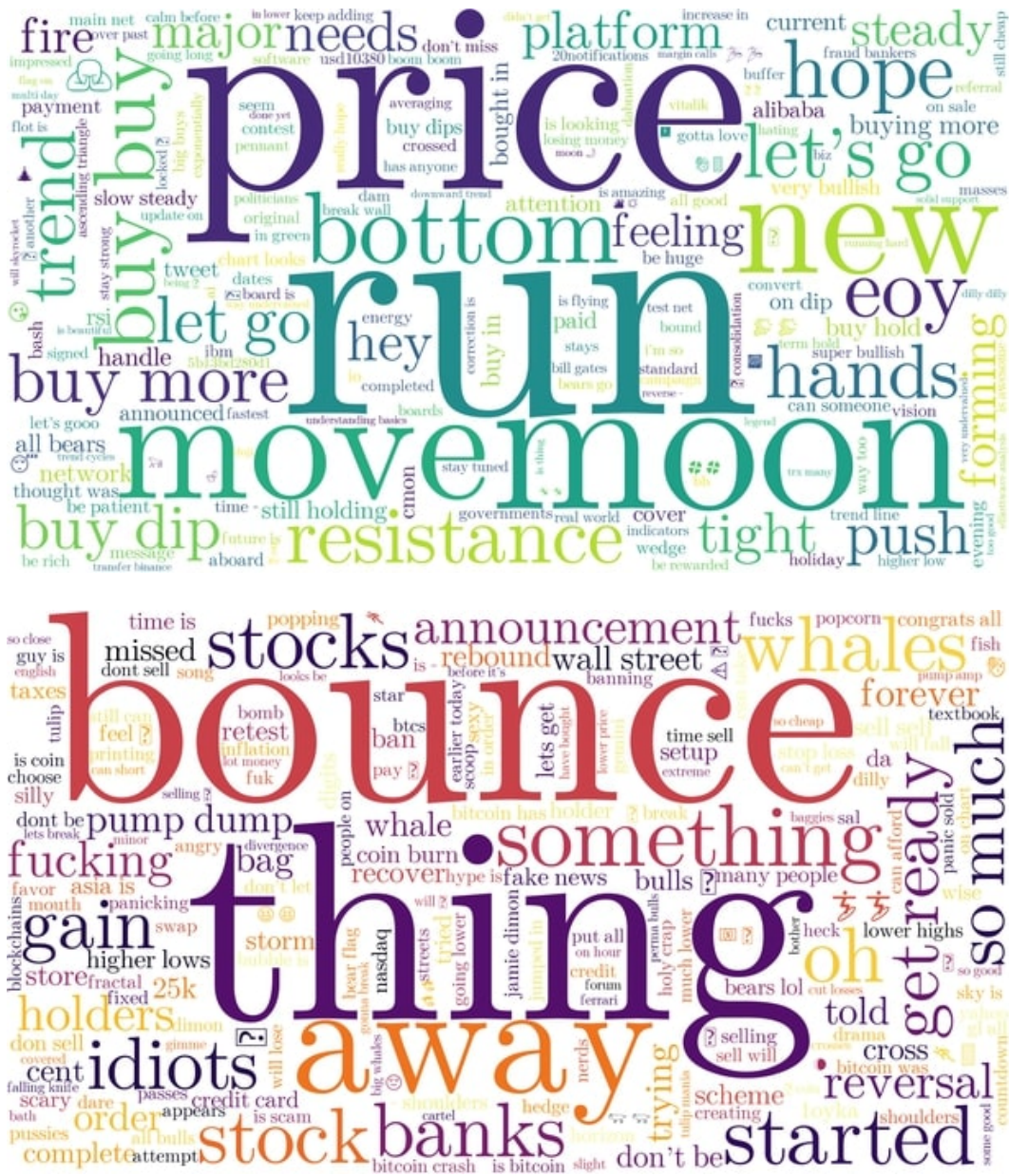


Figure 6: Word clouds for bullish (top) and bearish (bottom) context-specific terms augmenting the seed lexicon

As a baseline case, we also calculate $scores_m$ and the corresponding $sent^{seed}$ using only the sentiment weights of the seed terms.

Another approach to constructing the sentiment index is to use a trained RNN model to predict sentiment labels of unlabeled messages which constitute about 60% of the StockTwits' messages' dataset. We use the LSTM setup with pre-trained Word2Vec embeddings shown previously to have the best performance for this purpose. Aggregated sentiment is constructed in the following way: we obtain the number of bullish and bearish messages on day t : M_t^{Bu} and M_t^{Be} , respectively. Each quantity includes both known and predicted sentiments of each class. Then we test 2 sentiment measures calculated from M_t^{Bu} and M_t^{Be} :

1. as a logarithmic rate of change of the difference between the number of bullish and bearish

messages $\widetilde{D}_{M_t} = D_{M_t} + D_{adj}$ on a day t , where $D_{M_t} = M_t^{Bu} - M_t^{Be}$, $D_{adj} = \left| \min_t D_{M_t} \right| + 1$ is an adjustment to avoid non-positive values:

$$sent_t^{lstm1} = \log \left(\frac{\widetilde{D}_{M_t}}{\widetilde{D}_{M_{t-1}}} \right), \quad (20)$$

2. as an alternative "bullishness" measure proposed by Antweiler and Frank (2004)

$$sent_t^{lstm2} = \log \left(\frac{1 + M_t^{Bu}}{1 + M_t^{Be}} \right). \quad (21)$$

Many more alternative ways to construct the sentiment index can be considered such as taking the maximum score rather than averaging sentiment scores across messages. Another way would be to also come up with a different linear/non-linear function of various aggregate measures of bullish/bearish messages. We leave these endeavors for future research.

In the next section, we perform econometric analysis of predictability of the cryptocurrency index CRIX using three sentiment measures defined above.

5 Crypto sentiment index and asset return predictability

Figure 7 shows the time series of the crypto sentiment index and the log-returns of CRIX over time. We would like to find out if the sentiment index has value regarding predictability of the cryptocurrency index returns.

5.1 Return predictability of crypto assets

We consider the standard predictive regression model which is formulated as follows:

$$r_{m,t+1} = \alpha + \beta \cdot sent_t + \phi \cdot Z_t + \epsilon_{t+1}, \quad (22)$$

where $r_{m,t+1}$ is the log return of CRIX, Z_t is a vector of control predictors which includes the logarithm of message volume ($MsgVol$) and the moving average of $r_{m,t}$ (MA) as suggested by Detzel et al. (2018). Furthermore, $sent_t$ is one of the sentiment measures defined by $sent^{seed}$, $sent^{expand}$, $sent^{lstm1}$ and $sent^{lstm2}$. Our main goal is to test the significance of β , after controlling for Z_t .

Table 6 shows the results of in-sample predictive regressions using the sample period from August 2014 to May 2018. Three sentiment measures quantified by $sent^{seed}$, $sent^{expand}$, $sent^{lstm1}$ and $sent^{lstm2}$ are reported. By comparing their significance, we observe that the sentiment indices built using domain-specific information all make a more significant contribution to CRIX return predictability compared to the domain-general $sent^{seed}$ measure built using only the seed lexicon.

The best significance results are achieved for $sent^{expand}$, the measure constructed using the expanded lexicon. It appears that including domain-specific information in the form of lexicon

expansion contributes information relevant for return prediction. Chen et al. (2018) discover that topics contributing to predictability are related to the financial aspects such as market activities and transactions, whereas discussions about technology-related topics (blockchain, mining, wallets) on Reddit are less informative regarding short-run predictions.³

Research by Chen et al. (2018) confirms the value of StockTwits for market return predictability. Topics discussed there reflect investors' outlook from global and industrial perspectives, which is relevant for future price dynamics. With sentiment being considered, we find that explanatory power of the technical indicators proposed by Detzel et al. (2018) has disappeared; message volumes of StockTwits, as a proxy for market attention, cannot provide additional information with regards to return predictability.

Out-of-sample predictive performance is a natural robustness check for in-sample predictions. A model with good in-sample predictive performance may suffer from overfitting, so out-of-sample performance is a useful diagnostic tool. Welch and Goyal (2007) produced a comparison between a model exclusively relying on the historical average and a model with exogenous predictors. They proposed an out-of-sample R-squared statistic to test this comparison:

$$R_{oos}^2 = 1 - \frac{MSE_A}{MSE_N}, \quad (23)$$

where MSE_A is the mean squared error of the model with exogenous regressors, MSE_N is the mean squared error of the historical average model. Obviously, if the augmented model yields an MSE similar to that of the historical mean model, the additional information gain from the predictive model is limited. We therefore look for predictive models whose mean square errors are smaller than those obtained from the historical average-based models.

Given an estimation window of 1 year, out-of-sample forecasts for the future return are recursively computed, see Spiegel (2008). The out-of-sample period is from August 2015 to May 2018. The results of out-of-sample forecast evaluation for each in-sample setup are documented in Table 6. The models with sentiment predictors $sent^{lstm1}$ and $sent^{expand}$ yield out-of-sample R-squared results of 0.064 and 0.055, respectively, whereas the model using the control variables and $sent^{expand}$ produces an out-of-sample R-squared of only 0.014. Therefore we can conclude that sentiment-augmented models predominantly perform better than the baseline both in- and out-of-sample.

5.2 Return predictability of traditional assets

To answer the question whether predictability of returns given sentiment is exclusive to cryptocurrencies, we take stock price returns of Apple Inc. (AAPL) as a comparison benchmark for its representative role on the social media and stock markets. Discussions about Apple in the social media are extremely popular as it draws a lot of attention from investors and users of Apple-made devices. For the purpose of comparison, we collect 449,761 relevant messages from 26,521 distinct users who discuss AAPL, posted between May 2017 and January 2019. Overall, 133,316 messages are classified as bullish (29.6%) and 48,186 as bearish (10.7%), while the remaining messages are unclassified. Similar to the case of cryptocurrencies, the imbalance between the numbers of positive and negative messages shows that online investors are optimistic generally.

³Reddit is a generic message board; a message board dedicated only to financial markets, covers a wider number of topics related to cryptocurrencies including discussions about cryptocurrency technology such as the blockchain.

Dependent variable	$r_{m,t+1}$				$r_{AAPL,t+1}$
$sent^{stm1}$	0.355 (0.014)				
$sent^{stm2}$		0.340 (0.017)			
$sent^{expand}$			0.314 (0.005)		
$sent^{seed}$				0.272 (0.082)	0.128 (0.134)
$MsgVol$	-0.036 (0.528)	-0.030 (0.586)	0.032 (0.460)	-0.006 (0.900)	-0.308 (0.033)
MA	Yes	Yes	Yes	Yes	Yes
R-squared	0.788	0.760	0.926	0.627	1.729
Out-of-sample R-squared	0.064	-0.133	0.055	0.014	-0.818

Table 6: Predictive ability comparisons

The table reports the results for predictive regression estimation for CRIX and AAPL log-returns. The control variables include the returns' moving average (MA) and message volume ($MsgVol$). $sent^{expand}$, $sent^{stm1}$, $sent^{stm2}$ are defined in (19), (20) and (21), respectively. The p -values are reported in parentheses and have been computed using Newey-West standard errors. The sample period is 2014-08-01 – 2018-05-15. The value of the out-of-sample R-squared is shown in percent.

We use the general finance-themed seed lexicon from Renault (2017) to quantify the sentiment toward AAPL and also test Apple's stock return predictability both in-sample and out-of-sample. As follows from Table 6, sentiment is not as informative compared to what we find for cryptocurrency. The message volume variable $MsgVol$ of AAPL has an effect of reverse predictability, that is, higher message volume today means lower return tomorrow. Here high attention, reflected by higher message volume, triggers behavioral overreaction. Although the value of R-squared is relatively higher, it is caused by high explanatory power of message volume instead of sentiment itself. At the same time, out-of-sample performance is worse, which indicates adverse informational contribution of the sentiment variable, compared to the historical average benchmark.

5.3 Sentiment and market volatility dynamics

Market fluctuations are often accompanied, preceded or followed by sharp increases in discussion intensity on relevant online media. Therefore we can set up a hypothesis that sentiment extracted from StockTwits' microblogging users discussing cryptocurrency-related topics may carry predictive value relative to the volatility of the underlying market. To incorporate distilled sentiment into the dynamics of the variance process, we use squared sentiment measure as an exogenous variable in a GARCH framework.

The sentiment-driven conditional variance dynamics are modeled as the integrated GARCH (IGARCH) approach, Engle and Bollerslev (1986), given the non-stationary nature of the cryptocurrency variance process. The property of stationarity in the second moment is violated due to the presence of permanent shocks. The IGARCH model is specifically suited to address the issue of highly persistent variance.

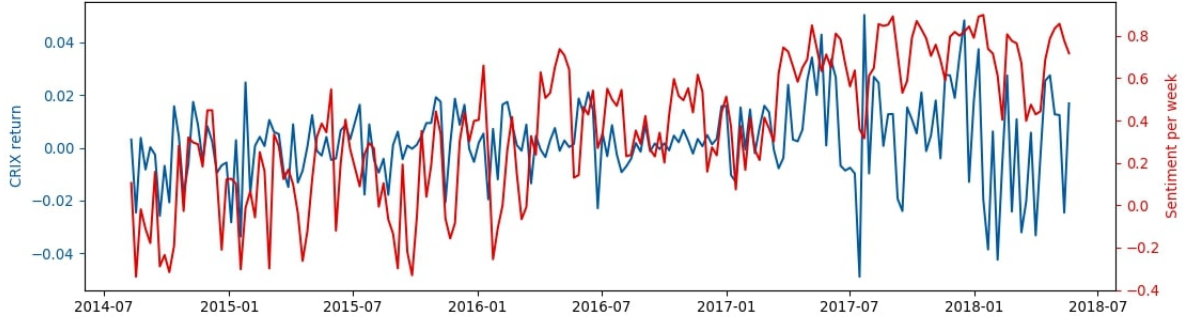


Figure 7: Co-movement between CRIX return and sentiment index (weekly)
The sentiment measure constructed via RNN prediction defined in (20) is demonstrated.

The IGARCH setup with an exogenous variable X_t can be formulated as follows:

$$e_t = y_t - \mathbf{E}_{t-1}y_t, \quad (24)$$

$$e_t = Z_t\sigma_t, \quad Z_t \sim t(\nu),$$

$$\sigma_{t+1}^2 = \alpha e_t^2 + (1 - \alpha)\sigma_t^2 + \theta X_t, \quad (25)$$

where $0 < \alpha < 1$, \mathbf{E}_{t-1} is the expectation operator conditional on information available at $t - 1$, σ_t^2 represents the conditional variance of the process at time t , $t(\nu)$ refers to the zero-mean t distribution with ν degrees of freedom. The exogenous variable X_t here is the squared sentiment measure $(sent^{seed})^2$, $(sent^{expand})^2$, $(sent^{lstm1})^2$ or $(sent^{lstm2})^2$. The IGARCH(1,1) specification is chosen based on the BIC criterion.

As shown in Table 7, sentiment drives the conditional variance process only when sentiment measures accommodate the domain-specific information, and the findings exclusively hold for cryptocurrency. General finance-related sentiment performs worse for the crypto asset class. In other words, volatility fluctuations in this market are more attributed to domain-specific sentiment.

In Figure 7, the co-movements between the CRIX index returns and a sentiment index are shown. Figure 8 displays the dynamics of the conditional volatility of the CRIX index along with its absolute returns. The variance dynamics in the specification of (25) driven by sentiment ($sent_{expand}$) is very similar to the dynamics of the absolute return-based realized volatility of the cryptocurrency index. The crypto market experienced a number of large fluctuations starting from 2017 until the first quarter of 2018. The sentiment-driven volatility model is capable of capturing these fluctuations quite well.

6 Conclusion

In this paper, we study market sentiment of cryptocurrency investors and traders on the Stock-Twits platform. We apply machine learning methods such as recurrent neural networks (RNNs) to construct sentiment indices reflecting opinions of the cryptocurrency community on the market through time. Next, we integrate the newly built sentiment indices into predictive regressions for autoregressive mean and variance of the returns of the CRIX cryptocurrency index.

Coefficient	Estimate	Robust std	<i>p</i> -value
CRIX			
$X = (\text{sent}^{lstm1})^2$			
α	0.16754	0.02579	0.000
θ	0.00118	0.00049	0.019
ν	3.34760	0.19149	0.000
$X = (\text{sent}^{lstm2})^2$			
α	0.16953	0.02585	0.000
θ	0.00081	0.00034	0.016
ν	3.28249	0.19194	0.000
$X = (\text{sent}^{expand})^2$			
α	0.14926	0.01831	0.000
θ	0.00021	0.00009	0.027
ν	3.65245	0.21875	0.000
$X = (\text{sent}^{seed})^2$			
α	0.12015	0.02177	0.000
θ	0.00421	0.00336	0.192
ν	3.71465	0.20182	0.000
AAPL			
$X = (\text{sent}^{seed})^2$			
α	0.14032	0.10555	0.183
θ	0.00146	0.00134	0.277
ν	3.82638	0.57326	0.000

Table 7: Estimated coefficients of IGARCH(1,1) model

The robust version of standard errors (robust std) are based on the method of White (1982).

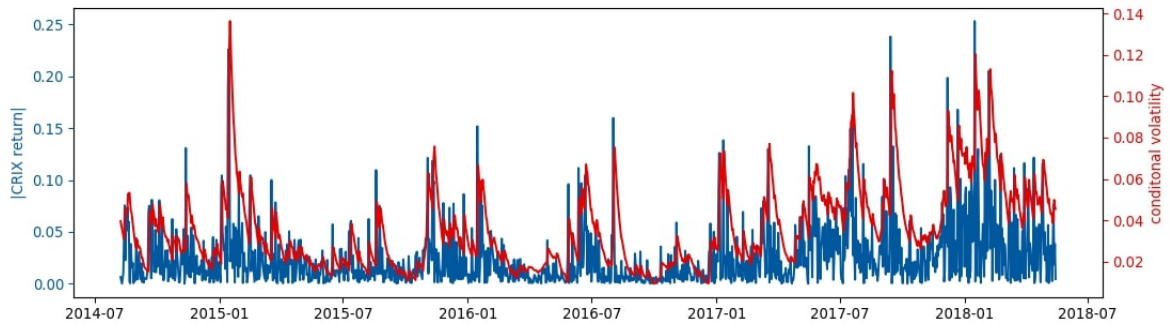


Figure 8: Sentiment-driven conditional volatility versus absolute return
The sentiment measure quantified by the extended lexicon defined in (19) is demonstrated.

We observe that the RNN LSTM setup with a pre-trained embedding layer gives the best predictive performance over other RNN setups with GRU or LSTM units, pre-trained or randomly initiated embeddings. This setup yields a higher recall with respect to bearish messages while also demonstrating high accuracy. Such an outcome is preferred for the imbalanced dataset with most of self-reported sentiment being positive, while bearish messages are under-represented. Errors in prediction of bearish messages are more costly because they directly transform into under-estimation of downside risk.

We set up two types of predictive regressions for the cryptocurrency index' log-return time series: for the autoregressive mean and variance. In the first case, adding the constructed sentiment index variables to the set of the regressors significantly contributes to predictability of the log-returns both in- and out-of-sample. In the second setup, we find that there is presence of unit root in the GARCH specification of cryptocurrency index' returns' volatility. We therefore use an IGARCH approach with squared sentiment as an additional predictor. We find that sentiment contribution to crypto volatility prediction is significant. The sentiment-driven volatility model is capable of capturing the actual fluctuations of absolute returns of the cryptocurrency index.

- Lerman, A. and Livnat, J. (2010). The new form 8-K disclosures, *Review of Accounting Studies* **15**(4): 752–778.
- Loughran, T. and McDonald, B. (2011). When is a liability not a liability? textual analysis, dictionaries, and 10-ks, *The Journal of Finance* **66**(1): 35–65.
URL: <http://dx.doi.org/10.1111/j.1540-6261.2010.01625.x>
- Mai, F., Shan, Z., Bai, Q., Wang, X. and Chiang, R. H. (2018). How does social media impact Bitcoin value? a test of the silent majority hypothesis, *Journal of Management Information Systems* **35**(1): 19–52.
- Mikolov, T., Chen, K., Corrado, G. and Dean, J. (2013). Efficient estimation of word representations in vector space.
URL: <https://arxiv.org/abs/1301.3781>
- Nassirtoussi, A. K., Aghabozorgia, S., Wah, T. Y. and Ngo, D. C. L. (2015). Text mining of news-headlines for FOREX market prediction: A multi-layer dimension reduction algorithm with semantics and sentiment, *Expert Systems with Applications* **42**(1): 306–324.
- Oliveira, N., Cortez, P. and Areal, N. (2016). Stock market sentiment lexicon acquisition using microblogging data and statistical measures, *Decision Support Systems* **85**: 62 – 73.
URL: <http://www.sciencedirect.com/science/article/pii/S0167923616300240>
- Pang, B., Lee, L. and Vaithyanathan, S. (2002). Thumbs up? sentiment classification using machine learning techniques, *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 79–86.
- Persio, L. D. and Honchar, O. (2016). Artificial neural networks architectures for stock price prediction: comparisons and applications, *International Journal of Circuits, Systems and Signal Processing* **10**: 403–413.
- Plakandaras, V., Papadimitriou, T., Gogas, P. and Diamantaras, K. (2015). Market sentiment and exchange rate directional forecasting, *Algorithmic Finance* **4**(1-2): 69–79.
- Renault, T. (2017). Intraday online investor sentiment and return patterns in the U.S. stock market, *Journal of Banking & Finance* **84**: 25 – 40.
URL: <http://www.sciencedirect.com/science/article/pii/S0378426617301589>
- Spiegel, M. (2008). Forecasting the equity premium: where we stand today, *The Review of Financial Studies* **21**(4): 1453–1454.
- Tetlock, P. C., Saar-Tsechansky, M. and Macskassy, S. (2008). More than words: quantifying language to measure firms' fundamentals, *The Journal of Finance* **63**(3): 1437–1467.
- Trimborn, S. and Härdle, W. K. (2018). CRIX an index for cryptocurrencies, *Journal of Empirical Finance* **49**: 107–122.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. and Polosukhin, I. (2017). Attention is all you need.
URL: <https://arxiv.org/abs/1706.03762>
- Welch, I. and Goyal, A. (2007). A comprehensive look at the empirical performance of equity premium prediction, *The Review of Financial Studies* **21**(4): 1455–1508.
- White, H. (1982). Maximum likelihood estimation of misspecified models, *Econometrica: Journal of the Econometric Society* pp. 1–25.
- Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. and Le, Q. V. (2019). XLNet: Generalized autoregressive pretraining for language understanding.
URL: <https://arxiv.org/abs/1906.08237>
- Yu, J. (2013). A sentiment-based explanation of the forward premium puzzle, *Journal of Monetary Economics* **60**(4): 474–491.

Appendix

Backpropagation through time (BPTT) for LSTM

We predict the probability of a bullish message y_T as $\hat{y}_T = \sigma(W_h h_T + b_h)$, where $h_T = o_T \odot \tanh(c_T)$, according to the above. A suitable loss function is the binary cross-entropy loss over samples n :

$$E_T = - \sum_n y_t^{(n)} \log(\hat{y}_T^{(n)}) + (1 - y_t^{(n)}) \log(1 - \hat{y}_T^{(n)}). \quad (26)$$

In the current case we deal with the many-to-one LSTM architecture: we predict a single value \hat{y}_T for each sequence, so the backward pass is simplified to the last time point T only compared to the backward pass for a many-to-many architecture.

We need to update $W_o, W_i, W_f, W_c, W_h, U_o, U_i, U_f, U_c, b_o, b_i, b_f, b_c, b_h$ via a gradient descent approach by updating their gradients along the input sequence x_1, \dots, x_T . We can start by deriving the backward pass equations for sample n and then apply stochastic gradient descent to update the parameters for all n .

Using equations (1) - (6), the backward pass equations are as follows:

$$\frac{\partial E_T^{(n)}}{\partial W_o} = \sum_t \delta o_t \odot o_t \odot (1 - o_t) \otimes x_t, \quad (27)$$

$$\frac{\partial E_T^{(n)}}{\partial W_i} = \sum_t \delta i_t \odot i_t \odot (1 - i_t) \otimes x_t, \quad (28)$$

$$\frac{\partial E_T^{(n)}}{\partial W_f} = \sum_t \delta f_t \odot f_t \odot (1 - f_t) \otimes x_t, \quad (29)$$

$$\frac{\partial E_T^{(n)}}{\partial W_c} = \sum_t \delta \tilde{c}_t \odot (1 - \tilde{c}_t^2) \otimes x_t, \quad (30)$$

$$\frac{\partial E_T^{(n)}}{\partial U_o} = \sum_t \delta o_t \odot o_t \odot (1 - o_t) \otimes h_{t-1}, \quad (31)$$

$$\frac{\partial E_T^{(n)}}{\partial U_i} = \sum_t \delta i_t \odot i_t \odot (1 - i_t) \otimes h_{t-1}, \quad (32)$$

$$\frac{\partial E_T^{(n)}}{\partial U_f} = \sum_t \delta f_t \odot f_t \odot (1 - f_t) \otimes h_{t-1}, \quad (33)$$

$$\frac{\partial E_T^{(n)}}{\partial U_c} = \sum_t \delta \tilde{c}_t \odot (1 - \tilde{c}_t^2) \otimes h_{t-1}, \quad (34)$$

$$\frac{\partial E_T^{(n)}}{\partial W_h} = h_T \cdot (\hat{y}_T - y_T). \quad (35)$$

where $\delta o_t = \partial E_T^{(n)} / \partial o_t$, $\delta i_t = \partial E_T^{(n)} / \partial i_t$, $\delta f_t = \partial E_T^{(n)} / \partial f_t$, $\delta \tilde{c}_t = \partial E_T^{(n)} / \partial \tilde{c}_t$ are found as follows

using the chain rule:

$$\begin{aligned}\delta o_t &= \delta h_t \odot \tanh(c_t), \\ \delta i_t &= \delta c_t \odot \tilde{c}_t, \\ \delta f_t &= \delta c_t \odot c_{t-1}, \\ \delta \tilde{c}_t &= \delta c_t \odot i_t.\end{aligned}$$

In the above, we used the facts that $\partial \tanh(x)/\partial x = 1 - \tanh^2(x)$ and $\partial \sigma(x)/\partial x = \sigma(x)(1 - \sigma(x))$. Furthermore, $\partial E_T^{(n)}/\partial b_h = \hat{y}_T - y_T$; gradients for b_o, b_i, b_f, b_c, b_h are obtained by removing outer products with x_t and h_{t-1} from equations (27)-(34), respectively.

Finally, derivatives with respect to the memory state c_t , the hidden state h_t and the input x_t are found as follows:

$$\delta c_t = \delta h_t \odot o_t \odot (1 - \tanh^2(c_t)), \quad (36)$$

$$\delta c_t = \delta c_t + f_{t+1} \odot \delta c_{t+1}, \quad (37)$$

$$\delta h_{t-1} = U_f^\top \cdot \delta f_t + U_i^\top \cdot \delta i_t + U_o^\top \cdot \delta o_t + U_c^\top \cdot \delta \tilde{c}_t, \quad (38)$$

$$\delta h_{t-1} = \delta h_{t-1} + W_h \cdot (\hat{y}_{t-1} - y_{t-1}), \quad (39)$$

$$\delta x_t = W_f^\top \cdot \delta f_t + W_i^\top \cdot \delta i_t + W_o^\top \cdot \delta o_t + W_c^\top \cdot \delta \tilde{c}_t. \quad (40)$$

Subtle details of LSTM backward pass implementation are contained in (37) and (39). Unlike GRU, in LSTM framework gradients are propagated via both hidden h and memory c channels; (37) reflects the accumulation of gradients via the memory state which is responsible for the transfer of "memory" along the sequence x . In a more general many-to-many framework, (37) follows from the chain rule for a function of several variables; for the error function $E^{(n)} = \sum_t E_t^{(n)}$ along all t :

$$\begin{aligned}\frac{\partial E^{(n)}}{\partial c_t} &= \frac{\partial E_t^{(n)}}{\partial c_t} + \frac{\partial E_{t+1}^{(n)}}{\partial c_t}, \\ &= \frac{\partial E_t^{(n)}}{\partial c_t} + \frac{\partial E_{t+1}^{(n)}}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial c_{t+1}} \frac{\partial c_{t+1}}{\partial c_t}, \\ &= \delta c_t + \delta c_{t+1} \odot f_{t+1}.\end{aligned}$$

In fact, the first term disappears in the many-to-one architecture and only the second term from the last step $f_T \odot \delta c_T$ is propagated backwards from end to start of x .

Furthermore, (39) describes propagation of gradients via the hidden state. At each point t , both inputs from (38) and loss function (26) are taken. To arrive at δh_t , inputs from the current loss and next-step δh_{t-1} are added together. Clearly, in the many-to-one architecture, $\delta h_t = \delta h_{t-1}$ for all t except the last T and $\delta h_T = W_h \cdot (\hat{y}_T - y_T)$.

In a deep LSTM network, the gradient δx_t will be used to propagate errors down to lower hidden layers. Thus, extension to networks with more one one layer such as one shown in Figure 1, is straightforward.

Finally, we can apply a gradient descent algorithm such as $\theta = \theta - \alpha \delta \theta$ with a learning rate α and $\delta \theta = \{\delta W_o, \delta W_i, \delta W_f, \delta W_c, \delta W_h, \delta U_o, \delta U_i, \delta U_f, \delta U_c, \delta b_o, \delta b_i, \delta b_f, \delta b_c, \delta b_h\}$.