# Predicting the quality of user contributions via LSTMs

Rakshit Agrawal
University of California, Santa Cruz
156 High St., Santa Cruz, CA 95064, USA
ragrawa1@ucsc.edu

Luca deAlfaro
University of California, Santa Cruz
156 High St., Santa Cruz, CA 95064, USA
luca@ucsc.edu

## ABSTRACT

In many collaborative systems it is useful to automatically estimate the quality of new contributions; the estimates can be used for instance to flag contributions for review. To predict the quality of a contribution by a user, it is useful to take into account both the characteristics of the revision itself, and the past history of contributions by that user. In several approaches, the user's history is first summarized into a number of features, such as number of contributions, user reputation, time from previous revision, and so forth. These features are then passed along with features of the current revision to a machine-learning classifier, which outputs a prediction for the user contribution. The summarization step is used because the usual machine learning models, such as neural nets, SVMs, etc. rely on a fixed number of input features.We show in this paper that this manual selection of summarization features can be avoided by adopting machine-learning approaches that are able to cope with temporal sequences of input.

In particular, we show that *Long-Short Term Memory* (LSTM) neural nets are able to process directly the variable-length history of a user's activity in the system, and produce an output that is highly predictive of the quality of the next contribution by the user. Our approach does not eliminate the process of feature selection, which is present in all machine learning. Rather, it eliminates the need for deciding which features from a user's past are most useful for predicting the future: we can simply pass to the machine-learning apparatus all the past, and let it come up with an estimate for the quality of the next contribution.

We present models combining LSTM and NN for predicting revision quality and show that the prediction accuracy attained is far superior to the one obtained using the NN alone. More interestingly, we also show that the prediction attained is superior to the one obtained using user reputation as a feature summarizing the quality of a user's past work. This can be explained by noting that the primary function of user reputation is to provide an incentive towards performing useful contributions, rather than to be a feature optimized for prediction of future contribution quality. We also show that the LSTM output changes in a natural way in response to user behavior, increasing when the user performs a sequence of good quality contributions, and decreasing when the user performs a sequence of low-quality work. The LSTM output for a user could thus be usefully shown to other users, alongside the user's reputation and other information.

*OpenSym '16, August 17 - 19, 2016, Berlin, Germany*
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4451-7/16/08...$15.00
DOI: http://dx.doi.org/10.1145/2957792.2957811

## CCS Concepts

•**Computing methodologies → Machine learning approaches;** *Sequential decision making; Neural networks;*

## Keywords

Reputation; Machine Learning; Wikipedia; LSTM; Neural Networks

## 1. INTRODUCTION

In many collaborative systems, users can contribute content, and the content can be subsequently edited, rated, or commented upon by other users. The Wikipedia, Stack Overflow, Yelp, and Quora are examples in which the user contributions can be thus modified or rated. After a contribution has been present in the system for some time, it is possible to observer how other users have interacted with it, and thus, in these systems, it is very useful to be able to estimate the quality of a user contribution as soon as the contribution is entered, before other users have had the possibility of interacting with it. An estimate of contribution quality can be used to flag some contributions for review, as well as for producing initial rankings of new content. For the Wikipedia, there has been a large body of work on automated methods for detecting vandalism and flagging revisions for review [17, 18, 20, 5, 16, 14]. These methods generally rely on a mix of machine learning and natural language processing; an yearly competition (PAN) compares the performance of such detection methods.

To predict the quality of a contribution by a user, it is useful to take into account the past history of contributions by that user, as well as more generally the history of all activity by that user on the system. Indeed, several approaches take into account factors such as the number of contributions, their timing, and so forth [20, 1]. In these approaches, the past activity of a user is first summarized into a number of features, such as number of contributions, user reputation, time from previous revision, and so forth. These features are then passed along with features of the current revision to a machine-learning classifier, which outputs a prediction for the user contribution. The summarization step is necessary because the usual machine learning models, such as neural nets, support vector machines, or tree-based classifiers, rely on a fixed number of input features. This requires summarizing the variable-length history of user activity into a fixed feature set. Choosing which summary features to include is a trial and error process.

We show in this paper that this manual selection of summarization features can be avoided by adopting machine-learning approaches that are able to cope with temporal sequences of input. In particular, we show that *Long-Short Term Memory* (LSTM) neural nets [12, 9] are able to process directly the variable-length history of a user's activity in the system, and produce an output that is highly predictive of the quality of the next contribution by the user. LSTMs are to neural nets (NNs) what sequential are to combinatorial logic circuits: LSTMs include memory elements that can store information about the input sequence they have seen, in order to produce their output. Our approach does not eliminate the process of feature selection, which is present in all machine learning. Rather, it eliminates the need for deciding which features from a user's past are most useful for predicting the future: we can simply pass to the machine-learning apparatus all the past, and let it come up with an estimate for the quality of the next contribution.

Precisely, given a sequence of past contributions $w_1, w_2, \ldots, w_{n-1}$ by a user, and a current contribution $w_n$, we predict the quality of $w_n$ as follows. We extract from $w_1, w_2, \ldots, w_n$ the feature vectors $f_1, f_2, \ldots, f_n$, each feature vector corresponding to one contribution. The vectors $f_1, f_2, \ldots, f_n$ are fed to an LSTM, whose output is in turn fed to a standard neural net which has as inputs the LSTM output, alongside the feature vector $g$ for $w_n$. The NN then outputs the prediction for the quality of $w_n$. The reason for this architecture is that the contributions $w_1, w_2, \ldots, w_n$ have already occurred, and for them we can compute feature vectors that are much richer than we can compute for $w_n$: they can include, for instance, the number of positive or negative ratings of the contribution, how it was then modified by others, and so forth. Furthermore, this approach enables us to measure accurately how much information about the user's past activity is required to attain a precise prediction for the quality of the latest contribution. We will provide comparisons between different ways for training the LSTM and the NN for the task.

We show that the prediction accuracy attained by using both the LSTM and the NN is far superior to the one obtained using the NN alone. This is unsurprising, and confirms that the past history of user contributions is highly useful to predict the quality of future contributions, an observation already made, e.g., in [1, 20]. More interestingly, we show that the prediction attained is superior to the one obtained using user reputation as a feature summarizing the quality of a user's past work [1, 3]. This can be explained by noting that the primary function of user reputation is to provide an incentive towards performing useful contributions, rather than to be a feature optimized for prediction of future contribution quality.

We observed that in order to attain good precision in our predictions, using LSTMs with only one output (but possibly many memory cells) is almost as good as using LSTMs with many outputs. Most of the information about a user's past activity, in other words, can be summarized in a single floating-point number. In this sense, the output of single-output LSTMs can be regarded as a "learned" notion of user reputation. Indeed, we show that the LSTM output is highly predictive of the quality of future contributions by the user, and it changes in a natural way in response to user behavior, increasing when the user performs a sequence of good quality contributions, and decreasing when the user performs a sequence of low-quality work. The LSTM output for a user could thus be usefully shown to other users, alongside the user's reputation and other information.

The LSTM output differs however from a traditional notion of reputation in two respects. First, it is not explainable: the rules are encoded in the LSTM parameters, and there is no simple way to explain to users why their reputation increased or decreased by some amount when they perform an edit. Second, it is highly sensitive to recent history. In usual reputation systems, high-reputation users would need to do much damage before losing their reputation. The LSTM output is trained for predictability, and reflects changes in behavior patterns far faster.

While our results are presented in the context of the Wikipedia, we believe that our LSTM-based approach can provide a blueprint for applying machine learning to the activity of users in collaborative systems.

After an overview of previous work, we present in Section 3 the machine learning setup we use. Results are given in Section 5, and the features we use in the machine learning are detailed in Section 6.

## 2. RELATED WORK

The problem of predicting the quality of a newly-entered Wikipedia revision has been considered in detail by many authors: indeed, a competition was held to compare various approaches to the problem [6]. The best approaches of that competition were based on timing analysis of revisions [20], language features [16], and user reputation [1]; the three approaches were then unified in [3].

Long-Short Term Memory neural nets were introduced in [12], and have been applied to a long list of problems, including learning sequence timing [8], recognizing handwriting [11], and speech [10]; a good overview can be found in [9]. Their main advantage over recurrent neural nets (RNNs) is that the memory can be much more long-lived, as it is held in special memory elements with their own reset and application networks, rather than being simply forward-fed at each time step.

Reputation systems for the Wikipedia were proposed in [2], and extended to ways of measuring author contributions in [4]. The formula we use for computing the quality of an individual contribution (or edit) to the Wikipedia is derived from this work. Methods for computing the quality of Wikipedia articles on the basis of an analysis of the interaction between the contributors have been proposed and evaluated in [13]. A related approach based on the study of cooperation on each article has been presented in [22]. Lifecycle-based methods have also been applied to the problem of Wikipedia article quality estimation in [23]. The problem of estimating article quality has been studied in [7], which advocates the construction of information-quality models tailored to various categories of Wikipedia articles.

## 3. LEARNING FROM THE PAST

### 3.1 The Wikipedia Data

A page $p$ of Wikipedia evolves through a sequence of revisions $r_0, r_1^p, r_2, \ldots$, where each revision $r_i$ has author $a_i$, for $i > 0$. For each revision $r_i$, we have a set of data, including for instance the timestamp of $r_i$ and the comment entered as $r_i$ was submitted. We are interested in measuring the *quality* $q_i$ of revision $r_i$. For this, we follow the approach of [2, 4]. Let $d$ be a metric between revisions, so that $d(r_i, r_j)$ is the distance between $r_i$ and $r_j$. The metric $d$ can be obtained as a version of edit distance; see [2] for the details. Then, the quality of $r_i$ as measured from a later revision $r_j$ of the same page, for $0 < i < j$, is given by

$$q(r_i \mid r_j) = \frac{d(r_j, r_{i-1}) - d(r_j, r_i)}{d(r_{i-1}, r_i)}\ , \qquad (1)$$

where $r_{i-1}$ is the revision preceding $r_i$ on the page. The numerator of (1) represents how much closer the revision became to $r_j$ due to $r_i$; the denominator represents the amount of change involved in producing $r_i$. The ratio thus expresses the fraction of change in producing $r_i$ that goes in the direction of the future $r_j$; in other words, how much the author of $r_j$ agrees with the change $r_{i-1} \to r_i$. The ratio is bounded between $-1$ and $+1$: when $q(r_i \mid r_j) = -1$, it indicates that the revision has been reverted, and when $q(r_i \mid r_j) = +1$, it indicates that the change $r_{i-1} \to r_i$ was preserved in $r_j$. We then compute a value $q_i$ for $r_i$ by averaging $q(r_i \mid r_j)$ over several $r_j$ following $r_i$ on the same page, taking care of considering only "reference" revision $r_j$ that are by a different author than $r_i$. This method for assessing revision quality was shown to closely associate with reversions and good revisions [4].

### 3.2 Neural nets and LSTMs

Our ingredients for learning are neural nets (NNs) and long-short term memory nets (LSTMs). An NN can be thought of as a learning element that takes a fixed-size vector of input features $\langle x_1 \cdots x_n \rangle$, and produces a fixed-size vector of output features $\langle y_1 \cdots y_m \rangle$, for some $n, m > 0$. The NN is trainable: by defining a *loss function* between the obtained output and a desired *target* output, and back-propagating the derivative of the loss, the parameters of the neural net can be tuned using gradient descent to minimize the loss, and the net can learn the desired function. In our work, we use NNs trained with the AdaDelta algorithm [24].

An LSTM takes as input a sequence of vectors $\langle x_1^1 \cdots x_n^1 \rangle, \ldots, \langle x_1^k \cdots x_n^k$, for some $k > 0$. Each input vector $\langle x_1^i \cdots x_n^i \rangle$ yields an output $\langle y_1^i \cdots y_m^i \rangle$, for $0 < i \leq k$, and the final vector $\langle y_1^k \cdots y_m^k \rangle$ produced once all the input has been consumed is taken to be the output of the LSTM in response to the sequence of the input vectors. The details can be found, e.g., in the overview [9]. Again, the output of the LSTM can be trained by backpropagation; also in this case we use AdaDelta for the backpropagation step. The backpropagation is more involved, as it is necessary to backpropagate across the sequence of input vectors (an example of backpropagation through time [19]). LSTMs have internal memory elements, so that their output can depend on inputs at any time in the sequence of input vectors. We note that an LSTM with $M$ memory elements will in general produce $M$ outputs, one per memory cell. However, it is possible to discard some of those outputs, measuring the loss of the LSTM only according to the difference between a subset of output (possibly only one) and their target values. We will often talk about "LSTMs with one output": the LSTMs we use can have multiple memory cells, but only one of their outputs is used as the overall output and trained with respect to the target values.

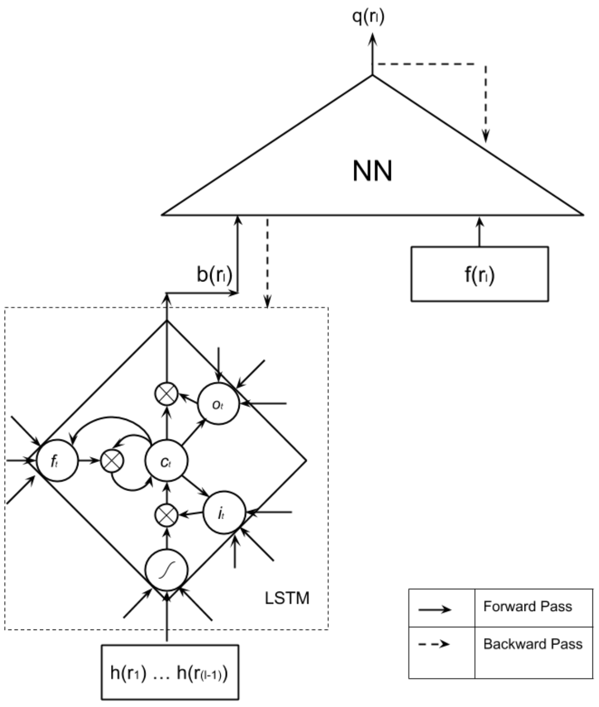

**Figure 1: Model using an LSTM combined with Neural Net and trained in chain. Error backpropagates through both Neural Net and LSTM. Output from Neural Net represents the final predicted quality.**

### 3.3 Applying NNs and LSTMs to user activity

We will apply neural nets and LSTMs to the work done by users across the Wikipedia pages they edited. Let $r_1, \ldots, r_l$ be the revisions entered by a user; note that these revisions, unlike those considered in Section 3.1, do not in general belong to the same page. For each of these revisions, we can produce a feature vector for consideration in machine learning; the vector can contain features related for instance to the number of words added, the time at which the revision was added, and so forth. The details of the features we extract are not of primary importance; a list of the features can be found in Section 6.

We distinguish, however, two different types of feature vectors for each revision $r$: the *foresight* feature vector $f(r)$, and the *hindsight* feature vector $h(r)$. The foresight feature vector $f(r)$ includes features known up to the time immediately after $r$ has been performed. These features include, for instance, the text composition of $r$ itself, but not how $r$ is subsequently altered by other users. The hindsight feature vector $h(r)$ includes all features of the foresight vector, as well as features that can be measured about a revision only after some time has passed, and most importantly among them, the quality of $r$.

## 3.4 Strategies for learning from the past

We have experimented with two strategies for learning from the user behavior history. Let $n$ be the number of features in a hindsight vector, and $m < n$ the number of features in a foresight vector.

### 3.4.1 Training the LSTM alongside the NN

The first strategy is illustrated in Figure 1. We build an LSTM that can take as input a sequence of hindsight feature vectors. The output of the LSTM is then fed to a NN that has as input the LSTM output, alongside a single foresight feature vector, and only one output. For a user with edit history $r_1, \ldots, r_l$, we feed the hindsight vectors $h(r_1), \ldots, h(r_{l-1})$ to the LSTM. Then, we feed the output of the LSTM along with the foresight vector $f(r_l)$ to the NN. We train this LSTM-NN composite using as target for the NN the quality $q(r_l)$ of revision $r_l$ (the LSTM is then trained by backpropagating the loss derivative through the NN, as in any multi-stage neural model).

In this way, we train the NN to give as output its best guess of the quality of the revision $r_l$. In turn, the back-propagation through the NN will cause the LSTM to try to provide the most helpful features about the past history of a user in order to predict the quality of a newly-made revision. If we restrict the LSTM to have only one output feature, that feature will consist in the real-valued variable that contains the most informative summary about a user's previous work, for the purpose of predicting the quality of a future revision.

### 3.4.2 Training the LSTM in isolation

The second strategy is illustrated in Figure 2. In this strategy, we build an LSTM that can take as input the sequence of hindsight vectors corresponding to $r_1, \ldots, r_{l-1}$, as before. The LSTM has one output only, which is directly trained according to the quality $q(r_l)$ of the *next* revision $r_l$. Thus, the LSTM is directly trained to be a predictor of quality of work for a user.

### 3.4.3 Training data

We construct the training sequences as follows. We select users randomly on the Wikipedia, and for each user, we consider the sequence $r_1, \ldots, r_N$ of all work performed by the user, ending with the last revision $r_N$ for which we are still able to measure the quality. We then consider the subsequences $[r_1]$, $[r_1, r_2]$, $[r_1, r_2, r_3]$, $\ldots$, $[r_1, r_2, \ldots, r_N]$, and for each of these $N$ sequences $[r_1, r_2, \ldots, r_l]$ with $1 \leq l \leq N$, we apply the training methods above. In this way, we use a single sequence of user work to train our models to predict the quality of future revisions at all points along that sequence. This increases the amount of training data available to us.

## 3.5 Comparing LSTMs with user reputation

In order to compare the effectiveness of LSTMs and user reputation as ways of summarizing the past, we compute a notion of reputation which essentially coincides with that of [2, 1]. Let $r_1, r_2, \ldots, r_n$ be the revisions performed by a user, and let $q_1, q_2, \ldots, q_n$ be their qualities. Let also $d_i$ be the distance between revision $r_i$ and the immediately preceding revision on the same page, for $1 \leq i \leq n$; in other words, $d_i$ is the amount of work performed by the user in producing $r_i$. The reputation of the user after producing $r_1, r_2, \ldots, r_n$ is then computed as:

$$\sum_{i=1}^{n} q_i \sqrt{d_i}$$

where the square root is used in place of an identity function to boost the weight of short revisions, following [2]. We note that our experimental results would hold also for the alternative form of reputation $\sum_{i=1}^{n} q_i d_i$.

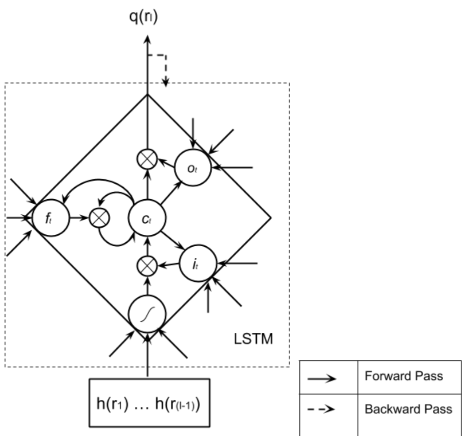

**Figure 2: Model for using only LSTM and passing all revision features to it. Output from the LSTM alone represents predicted quality**

## 4. WIKIPEDIA DATA

For our experiments, we used data from the English Wikipedia [21]. The data was collected by randomly selecting a large number of authors on the Wikipedia using API provided by the MediaWiki Foundation [15]. For each randomly selected author, we collected a list of their contributions (edits) on Wikipedia starting from the very first. Since the paradigm of user participation might change drastically over time, we limited our users to the ones that started contributing to Wikipedia from January 1st, 2012. We did not consider any users who started contributing before that date. For our experiments, we used upto 50 contributions by each user starting from the very first. MediaWiki API provides an interface to obtain the content of any revision along with some meta data including pageid, parent revision, timestamp, revision comment, etc. As detailed in section 6, we extracted a large number of features directly from the content, and used meta data to extract additional features.

In order to obtain this data from Wikipedia, we contacted the MediaWiki API using the REST interface. This API provides different types of queries for properties and lists. To get a random list of users, we contacted the *list* interface. Using each author's username from the list then, we contacted the *list* interface holding *user contributions*. This provided us with a list per user for the contributions containing each revision's ID in the Wikipedia database. Then using this ID for each revision, we contacted the *properties* interface for *revision*. This provided us with revision's content and associated meta-data. In the process of extracting

**Table 1: Evaluation of Different Learning Models**

| | Precision | Recall | F-score |
|---|---|---|---|
| NNet only | 0.655 | 0.291 | 0.403 |
| Reputation + NNet | 0.659 | 0.744 | 0.699 |
| LSTM in isolation | 0.769 | 0.788 | 0.778 |
| LSTM-NNet combined | 0.795 | 0.821 | 0.808 |
| Trained LSTM with NN | 0.735 | 0.712 | 0.723 |

features, we also needed to compare a revision with several other revisions on the same page. The *properties* interface for *revisions* allowed us to fetch consecutive revisions on a page along with added parameters.

This data containing list of authors and revisions was stored locally in a database. Before storing a revision into the Database, we performed an entire round of feature collection per revision and stored the revision along with its measured features into the database. For purpose of training and testing our learning model, we generated JSON files consisting of normalized feature data from this database, and split it randomly to get separate training and test datasets.

## 5. RESULTS

As explained above in Section 3.4, we implemented different training approaches on our data targeted at predicting the quality, $q(r_l)$ of revision $r_l$. Along with the two models mentioned, we also implemented two additional combinations where we, (a) used a pre-trained LSTM as a function while training a neural network, and (b) where we just trained a neural net by sending each revision as an individual entry. In our process, we used over 5000 random users from Wikipedia. For each user, we collected up to their first 50 contributions(revisions) on Wikipedia. Further we also ensured to balance the inputs to our learning models in order to maintain equivalent probability of labels. The quality $q(r_l)$ of each revision is a continuous value in the range $[-1, 1]$. To evaluate our models, we categorized this value into high and low quality by taking the values of $q(r_l) > 0$ to be high quality and $q(r_l) < 0$ to be low quality. These classification labels (high and low) were then used to measure precision, recall and F-score for each training model. A summary of our results is provided in Table 1. Among the methods we used to predict revision quality, the first two methods are trained with history of user through an LSTM and provide useful value representing user's history. The third method uses a pre-trained LSTM to generate an input from user's history while training the Neural Net. The fourth method simply uses a Neural Net and considers every revision individually, with no past information and no LSTM use. In comparison with reputation of user, we also tested an additional model where instead of deriving history using LSTM, we measured user's reputation as described in Section 3.

### 5.1 Training the LSTM in isolation

The first history based model we used was a separate LSTM where the predicted quality $q(r_l)$, for revision $r_l$ relies entirely on user's edit history $r_1, \ldots, r_{l-1}$. This model therefore, takes as input the hindsight vectors $h(r_1), \ldots, h(r_{l-1})$ and does not many any use of foresight vector $f(r_l)$ for revision $r_l$. As discussed in Section 3.4, this LSTM has only one output and we trained it using quality $q(r_l)$ as the target. After training for multiple iterations, the LSTM produced an average F-score of around 0.77 on the test data. This result, in comparison to the Neural Net model discussed later shows significant improvement in the prediction for quality of a revision. The trained LSTM model for predicting quality $q(r_l)$ can be therefore represented as :

$$H(r_l) = h(r_1), \ldots, h(r_{l-1}) \tag{1}$$

$$q(r_l) = LSTM(H(r_l)) \tag{2}$$

### 5.2 LSTM and NN combined model

The primary model in our experiments, as explained in Section 3.4 as well, constituted of both the Neural Net and LSTM connected to each other and trained together. In this model, for each user $i$, we passed the user's edit history $h(r_1), \ldots, h(r_{l-1})$ through an LSTM, whose output $b(r_l)$ was directly sent to the Neural Network. From the $m$ output values produced by LSTM, we used only one output and passed it to the Neural Net. This output is a measure for user's history on Wikipedia. Along with this output, we fed the foresight vector $f(r_l)$ to the Neural Net. This LSTM-NN composite model was trained using quality $q(r_l)$ for revision $r_l$ as target for the Neural Net. LSTM was trained by backpropagating the loss derivative through the Neural Net. This model trained the LSTM along with the Neural Net and therefore the LSTM was providing a representation of user's history instead of predicting the quality itself. The value predicted by the LSTM in this model was further used along with foresight features $f(r_l)$ of revision $r_l$ and their combined relation predicted quality. Based on our results we discovered that this value $b(r_l)$ generated in the combined model was inversely proportional to the predicted quality. On an average, through this model, we achieved an F-score of around 0.78 with it reaching even up to 0.808 in some cases. The complete model generated here is a combination of the trained LSTM followed by a trained Neural Net for providing the final predicted quality. It can be represented as:

$$H(r_l) = h(r_1), \ldots, h(r_{l-1}) \tag{1}$$

$$b(r_l) = LSTM(H(r_l)) \tag{2}$$

$$q(r_l) = NNet(b(r_l), f(r_l)) \tag{3}$$

### 5.3 LSTM trained in isolation with Neural Net

With the LSTM trained for predicting quality of a revision, we tried another model that used the same trained LSTM and used it as an input generator for a Neural Net (Figure 3). In order to predict quality $q(r_l)$, for revision $r_l$ of a user, we trained a Neural Net with it's input being the foresight vector $f(r_l)$ along with an output, $b(r_l)$ from the trained LSTM, derived by passing hindsight vectors $h(r_1), \ldots, h(r_{l-1})$ to it. In this model, we only trained the Neural Net and used the pre-trained LSTM as a function to derive value $b(r_l)$ from user's edit history. The Neural Net was trained with target $q(r_l)$ for revision $r_l$. Backpropagation in this model was also performed using AdaDelta. The primary difference exhibited by this model was the use of user's history through a separately trained model. Instead of using LSTM to provide a measure of user's past, we used the LSTM trained for predicting quality $q(r_l)$ itself in this

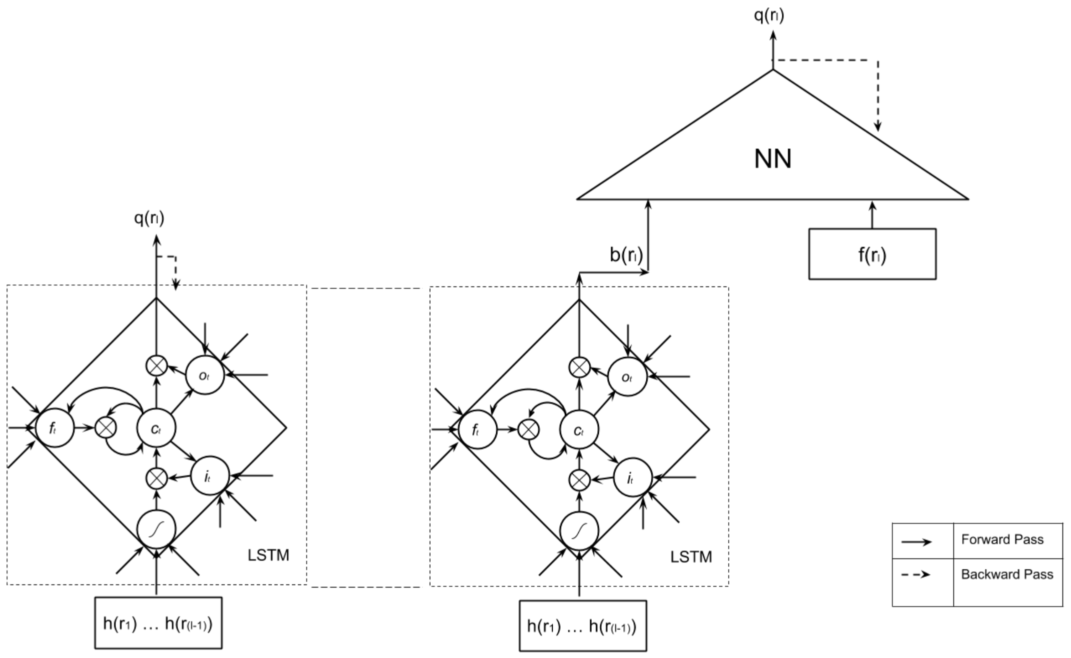

**Figure 3: Model using a trained LSTM to get history input along with revision's features into the Neural Network. Output from Neural Net represents the predicted quality**

model. Using this model, we achieved on an average, an F-score of around 0.72. The trained model for this combination can be represented as:

$$H(r_l) = h(r_1), \dots, h(r_{l-1}) \tag{1}$$

$$b(r_l) = LSTM_{\mathbf{T}}(H(r_l)) \tag{2}$$

$$q(r_l) = NNet(b(r_l), f(r_l)) \tag{3}$$

## 5.4 Neural Network only

In this model we tested the predictive power of a Neural Network by using only the foresight features, $f(r_l)$, of a revision $r_l$ as input data. For each revision $r_i$ in our dataset, we passed the foresight features $f(r_i)$ through the Neural Net, and compared the output with it's quality $q(r_i)$. We measured the loss $\mathcal{L}(r_i)$ for each revision and used it to perform backpropagation through the net. We used AdaDelta [24] for backpropagation We trained the Neural Net by repeating this process for multiple iterations. At each iteration we shuffled our data. After several variations in iterations, a well trained system generally produced an F-score ranging between 0.3-0.4 on the test data. The trained model generated from this approach can be represented as:

$$q(r_l) = NNet(f(r_l)) \tag{4}$$

## 5.5 Comparing LSTMs with user reputation

As discussed earlier in Section 3, we also evaluated our model against user reputation instead of LSTM output. In this model, for revision $r_l$ of user, we passed the foresight features $f(r_l)$ to the Neural Net along with the reputation $\mu$ of the user. The Neural Net was then trained over entire data using this model and it's result was observed in comparison to the result provided by LSTM based models. In this model, for a well trained Neural Net, the system generally produced an F-score around 0.68 on the test data which is less than our

## 5.6 Relevance of LSTM output

The output $b(r_l)$ discovered from the LSTM constituting user's history information is the primary entity we derive from history in our approach. We believe this value consolidates user's performance on Wikipedia into a significant value which, in association with revision $r_l$'s foresight vector $f(r_l)$, helps predict the quality $q(r_l)$ of revision $r_l$. As seen in the comparison between using Neural Net for each revision individually, and models using user's edit history, user's history value $b(r_l)$ creates a significant impact on accuracy of determining the revision's quality. We further tested this value $b(r_l)$ generated by the LSTM in LSTM-NN combined model and checked its impact on the resulting quality. As shown by multiple trials, the predicted quality $q(r_l)$ is inversely proportional to the value of $b(r_l)$ produced by the LSTM. Also we checked the monotonicity of this value and derived that $NN(0) > NN(b) > NN(1), \forall b \in (0, 1)$. An example of relation between history value $b(r_l)$ and predicted quality $q(r_l)$ can be seen in figure 4 Moreover, in most cases the value $b(r_l)$ generated by LSTM also shows a very strong correlation with many hindsight features, particularly the quality $q(r_{l-1})$ of previous revision.

These relations show that the value derived from LSTM is a useful combined representation of user's past. Figure 5 shows the changes in the output from LSTM in the isolated trained LSTM with changes in quality of user's revisions. Along with each LSTM output, we show the previous revision's quality. It can be seen here that the quality of previous revision displays high correlation with the output from LSTM. Parts (a) and (b) represent users with mostly high

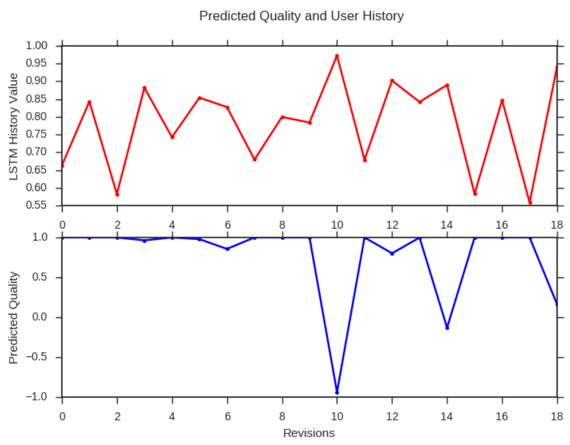

**Figure 4: Variations of LSTM output from LSTM-NN combined model with predicted revision's quality $q(r_l)$.**

quality edits, and (c) and (d) represent users with mostly low quality edits. In all four cases we can see strong relation between output and the quality.

In the LSTM-NN combined model, we observed that as the quality stays high, the value of output keeps on improving. Similarly, for a consistently low quality, the output in many cases tends to move in the opposite direction. This impact of consistent quality on the LSTM output adds support to the likeliness of this value being a measure of reputation. As mentioned earlier, this value alone is not explainable as reputation but it's growth is related to continued user performance. In all parts of Figure 5, the trend is similar. While a clear demonstration of LSTM output with quality is not achievable, in some cases we do observe the above discussed relation.

## 6. FEATURES AND QUALITY

In this section, we discuss the features that we extracted from our data and used them as entries in hindsight vectors $h(r_1), \ldots, h(r_{l-1})$ and foresight vector $f(r_l)$. In order to get these features, we used both content and meta-data of the revision. The features we used are divided into four categories:

### 6.1 Time Features

These features correspond to the time differences of revision under concern with other revisions on the page as well as with other revisions by the user. The values we derive here are:

#### *6.1.1 Time interval to previous revision on page*

This value refers to the interval from current revision on this page to previous revision on this page by any user. This is essentially the time difference between a revision and its parent revision. We consider this time difference to be an important feature since reversion of spam edits is performed very quickly on Wikipedia.

#### *6.1.2 Time interval to previous revision by user*

This value refers to the interval from user's previous edit on any page. This value is a measure of user's activity on Wikipedia in general. Since regular contributors usually establish time routines to work on edits and vandals might be active on multiple pages very quickly, it can be a helpful feature.

#### *6.1.3 Time interval to previous revision on page by user*

This value refers to the interval from user's last edit on this page. We believe this value is important to understand user's dedication to a particular page. Several contributors dedicate a large portion of their contributions to a particular page on Wikipedia.

### 6.2 Character Features

We obtained a large set of features from the content updated by the user in an edit. When a user performs an edit on a page, there can be either deletion or addition of content or movement of pieces. The character features help measure these character based changes performed in the current edit in comparison with its parent revisions. Features derived from this comparison are the following:

#### *6.2.1 Characters Added*

In order to measure the amount of contribution by the user on page, we need to get the amount of characters that have been added to the page. We compare the content with parent revision and gather the characters that have been added. The size then obtained is the amount of characters added in that revision. For the purpose of training our model, we normalize this value by using Z-value of the characters added clipped into the interval of $[-3, +3]$. Since our training model requires each value to be in the range $[0, 1]$, we transform the value thus obtained to be in this range. We believe that the size of contribution is a significant indicator of user's activity.

#### *6.2.2 Characters Removed*

Similar to characters added, we also measure the characters removed from previous revision. This value is obtained similarly by comparing with parent revision and then normalizing the value. The importance of this feature lies in the fact that a good contribution might need to remove a large amount of incorrect information, and a bad revision might attempt to delete a large amount of important text from the page.

#### *6.2.3 Spread of change within the page*

The distribution of change within a page can be an important feature as well. For this purpose, we measure the spread of changes on a page. The value is obtained by getting a list of all positions where changes (addition, deletion or movement) have been performed, and then measuring the spread of change around the page. We measure this by calculating standard deviation of positions with changes and normalizing it by dividing by the size of page. We believe that an estimate of distribution of contribution on a page might be a good indicator of user's overall actions on that page.

#### *6.2.4 Position in page*

We also measure a weighted position of edits on the page. This value is weighted average of additions and deletions on the page weighted by the size of change at each position.

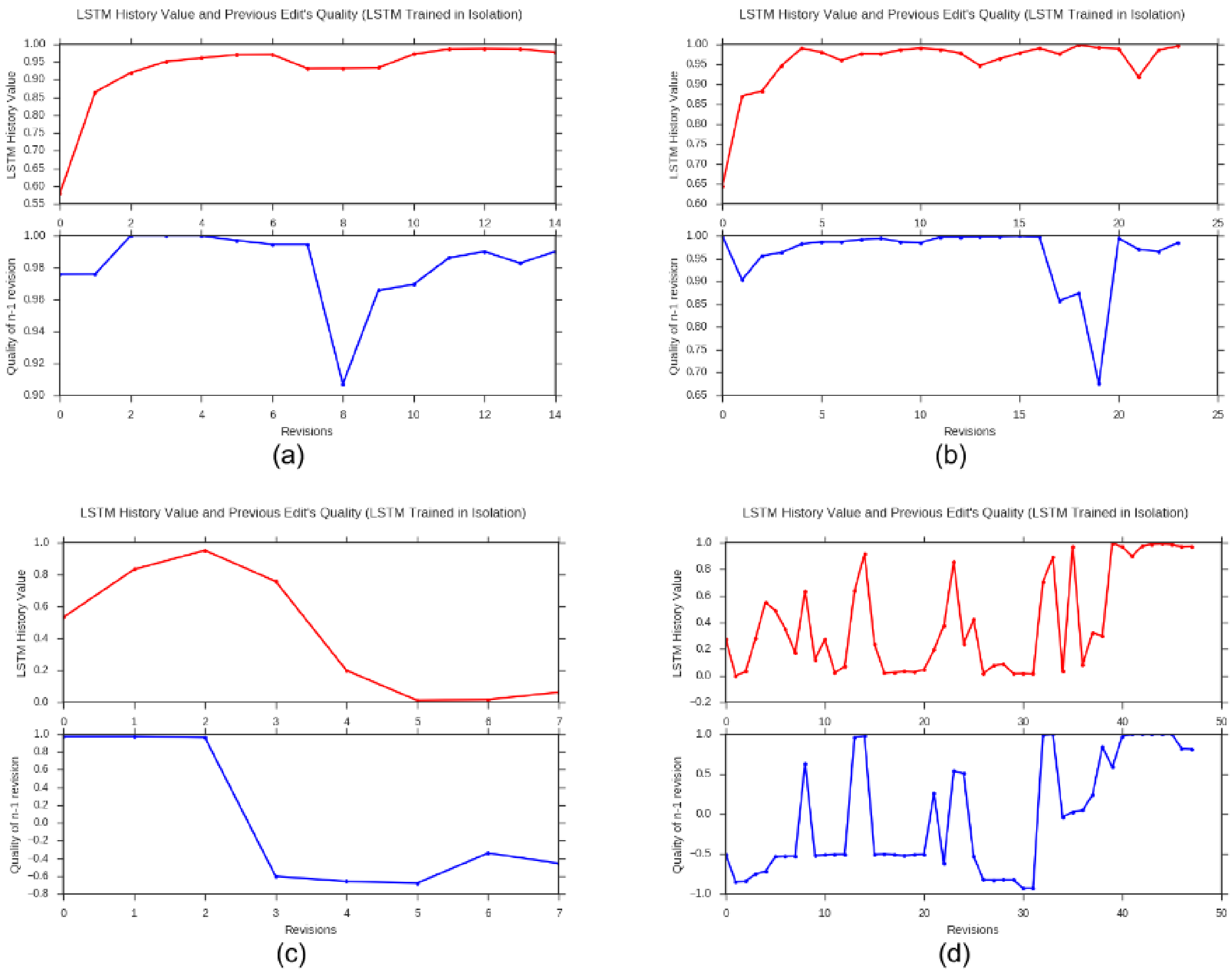

**Figure 5: Variations of LSTM output from LSTM Trained in Isolation with previous revision's quality. (a) and (b) represent users with mostly higher quality edits, (c) and (d) represent users with mostly low quality edits.**

This value gives an estimated position on the page where most of the action has been performed. Particular positions on a page can be indicators of positions requiring consistent edits or positions with increased vandal possibilities in some cases.

#### 6.2.5 *Upper Case/ Lower Case Ratio*

Apart from position and size based features, we also evaluate the contribution done by the user individually. From the text added by the user on a revision, we calculate the ratio of upper case and lower case letters. It is often possible that an incorrect edit might contain a larger proportion of upper case letters than generally required.

#### 6.2.6 *Digit/ Total Ratio*

Similar to the upper and lower case ratio, we also measure the ratio of digits in total contribution. Numerical values are often changed frequently on a page as they might be to update some values changing with time. Ratio of digits can therefore be helpful in estimating revision's quality.

### 6.3 Action Features

Along with the time and character features, we also calculate few features which determine the action performed by user rather than the resulting contribution. These features are about the act of writing a revision. The features we use are:

#### 6.3.1 *Revision Comment Length*

We measure the length of comment left by the editor on a revision. We assume that a genuine edit tends to have a longer revision comment than a vandal entry.

#### 6.3.2 *Time in Day*

Based on the timestamp of the edit, we use the time of the day when this edit was performed. The timestamp provides value in UTC. Therefore, this value can be helpful individually per user as it might indicate a pattern in user's activity. Dedicated users might work on edits during a particular time of the day. This value is normalized for use in our system by taking a sin of the Hour $H$. $t = \sin(2\pi H/24)$

#### 6.3.3 *Day of Week*

Calculated from the timestamp, we also use the day of week when this edit was made. An author might have a weekly pattern of contribution which might be helpful in generating a correlation with quality of revision.

### 6.4 Future Features

All the previously mentioned features are available right after an edit has been made. Those features compare revision to previous revisions and do not use any future values. Wherever available, we also measure some features for each revision relying on future revisions on the page. We calculate the following two future features:

#### 6.4.1 *Time to next revision on page*

This feature is similar to time features, but it measures the time interval from this revision to the next revision on page. We believe it can be an important feature since an incorrect edit tends to be reverted or modified very quickly while a good edit might stay unedited for a longer duration.

#### 6.4.2 *Measured Quality of Revision*

As described in Section 3.1, the quality $q(r_l)$ was an important part of a revision's hindsight features. Quality of each revision in user's history helped serve as an important feature in determining a useful value from the LSTM.

## 7. DISCUSSION

We have presented a machine-learning approach for predicting the quality of Wikipedia revisions that can leverage the complete contribution history of users when making predictions about the quality of their latest contribution. Rather than using ad-hoc summary features computed on the basis of user's contribution history, our approach can take as input directly the information on all the edits performed by the user. Our approach leverages the power of LSTMs (*long-short term memory* neural nets) for processing the variable-length contribution history of users. We present several approaches for combining LSTMs to process the revision history, and neural nets for combining the history information with features from the latest revision; we provided a detailed experimental comparison of the approaches. We show that our LSTM-based techniques are superior both to techniques that do not consider the contribution history of users, and to techniques that summarize each user's history into a single value of reputation.

We also show that the relevant information from a user's past that we extract via LSTMs can be quite small: one output (one floating-point number) contains most of the useful information. This single output is highly predictive of the quality of the next contribution by the user, and it can be understood as a notion of user reputation that is *learned,* rather than computed via a user-defined set of rules.

While we have presented our results in the context of Wikipedia, we believe that our joint use of LSTMs and neural nets can provide a general blueprint for applying machine-learning to the activity history of users in collaborative systems.

## 8. ACKNOWLEDGMENTS

This research has been supported in part by the NSF Award 1432690. We would also like to thank Vasileios Polychronopoulos for his valuable support and advice during the development of this paper.

## 9. REFERENCES

[1] B. Adler, L. de Alfaro, and I. Pye. Detecting wikipedia vandalism using wikitrust. *Notebook papers of CLEF*, 1:22–23, 2010.

[2] B. T. Adler and L. De Alfaro. A content-driven reputation system for the Wikipedia. In *Proceedings of the 16th international conference on World Wide Web*, pages 261–270. ACM, 2007.

[3] B. T. Adler, L. De Alfaro, S. M. Mola-Velasco, P. Rosso, and A. G. West. Wikipedia vandalism detection: Combining natural language, metadata, and reputation features. In *Computational linguistics and intelligent text processing*, pages 277–288. Springer, 2011.

[4] B. T. Adler, L. de Alfaro, I. Pye, and V. Raman. Measuring author contributions to the Wikipedia. In *Proceedings of the 4th International Symposium on Wikis*, page 15. ACM, 2008.

[5] S.-C. Chin, W. N. Street, P. Srinivasan, and D. Eichmann. Detecting Wikipedia vandalism with active learning and statistical language models. In *Proceedings of the 4th workshop on Information credibility*, pages 3–10. ACM, 2010.

[6] CLEF 2010 Labs and Workshops, M. Braschler, D. K. Harman, E. Pianta, and CLEF. *Abstracts of the notebook papers.* s. n.], S. l., 2010.

[7] G. De la Calzada and A. Dekhtyar. On measuring the quality of Wikipedia articles. In *Proceedings of the 4th workshop on Information credibility*, pages 11–18. ACM, 2010.

[8] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber. Learning precise timing with LSTM recurrent networks. *The Journal of Machine Learning Research*, 3:115–143, 2003.

[9] A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks.* PhD thesis, Technishe Universit\"at M\"unchen, 2012.

[10] A. Graves, N. Jaitly, and A.-r. Mohamed. Hybrid speech recognition with deep bidirectional LSTM. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278. IEEE, 2013.

[11] A. Graves, M. Liwicki, S. FernÃąndez, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(5):855–868, 2009.

[12] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[13] M. Hu, E.-P. Lim, A. Sun, H. W. Lauw, and B.-Q. Vuong. Measuring article quality in wikipedia: models and evaluation. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 243–252. ACM, 2007.

[14] S. Javanmardi, D. W. McDonald, and C. V. Lopes. Vandalism detection in Wikipedia: a high-performing, feature-rich model and its reduction through Lasso. In *Proceedings of the 7th International Symposium on Wikis and Open Collaboration*, pages 82–90. ACM, 2011.

[15] MediaWiki. Mediawiki API, 2006.

[16] S. M. Mola-Velasco. Wikipedia Vandalism Detection Through Machine Learning: Feature Review and New Proposals: Lab Report for PAN at CLEF 2010. *arXiv preprint arXiv:1210.5560*, 2010.

[17] M. Potthast, B. Stein, and R. Gerling. Automatic vandalism detection in Wikipedia. In *Advances in Information Retrieval*, pages 663–668. Springer, 2008.

[18] K. Smets, B. Goethals, and B. Verdonk. Automatic vandalism detection in Wikipedia: Towards a machine learning approach. In *AAAI workshop on Wikipedia and artificial intelligence: An Evolving Synergy*, pages 43–48, 2008.

[19] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[20] A. G. West, S. Kannan, and I. Lee. Detecting wikipedia vandalism via spatio-temporal analysis of revision metadata? In *Proceedings of the Third European Workshop on System Security*, pages 22–28. ACM, 2010.

[21] Wikipedia. Wikipedia, the free encyclopedia, 2004.

[22] D. M. Wilkinson and B. A. Huberman. Cooperation and quality in wikipedia. In *Proceedings of the 2007 international symposium on Wikis*, pages 157–164. ACM, 2007.

[23] T. WÃűhner and R. Peters. Assessing the quality of Wikipedia articles with lifecycle based metrics. In *Proceedings of the 5th International Symposium on Wikis and Open Collaboration*, page 16. ACM, 2009.

[24] M. D. Zeiler. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.