MELFORD: Using Neural Networks to Find Spreadsheet Errors

Rishabh Singh, Benjamin Livshits, and Benjamin Zorn

Microsoft Research

Microsoft Tech Report Number MSR-TR-2017-5

Abstract

Spreadsheets are widely used for financial and other types of important numerical computations. Spreadsheet errors have accounted for hundreds of millions of dollars of financial losses, but tools for finding errors in spreadsheets are still quite primitive. At the same time, deep learning techniques have led to great advances in complex tasks such as speech and image recognition. In this paper, we show that applying neural networks to spreadsheets allows us to find an important class of error with high precision. The specific errors we detect are cases where an author has placed a number where there should be a formula, such as in the row totaling the numbers in a column. We use a spatial abstraction of the cells around a particular cell to build a classifier that predicts whether a cell should contain a formula whenever it contains a number.

Our approach requires no labeled data and allows us to rapidly explore potential new classifiers to improve the effectiveness of the technique. Our classifier has a low false positive rate and finds more than 150 real errors in a collection of 70 benchmark workbooks. We also applied MELFORD to almost all of the financial spreadsheets in the EUSES corpus and within hours confirmed real errors that were previously unknown to us in 26 of the 696 workbooks. We believe that applying neural networks to helping individuals reason about the structure and content of spreadsheets has great potential.

1. Introduction

Spreadsheets[1] are used by hundreds of millions of users for many important tasks, including financial calculations. There has been a lot of research on spreadsheet errors that demonstrate the degree to which spreadsheets are used for important calculations and measure the degree to which many real, important spreadsheets contain errors [31]. Spreadsheet errors are in the news regularly, including recent news in 2016 that a 400 million dollar error caused by a spreadsheet was made in the analysis of SolarCity's valuation in its sale to Tesla [9].

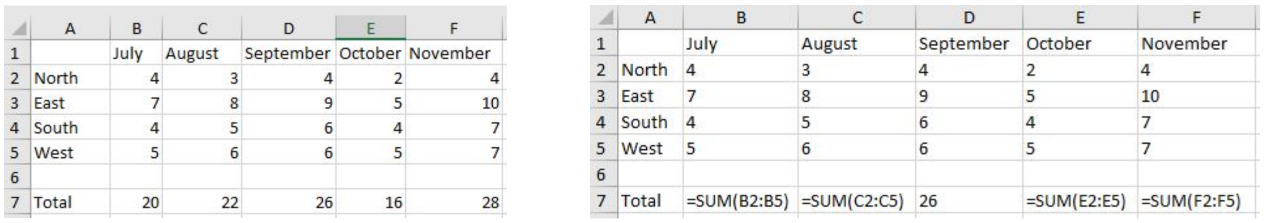
Despite widespread knowledge that spreadsheets often contain errors and that they can have consequences, the tools available to find such errors are relatively primitive. Many tools are based on applying rules that identify common anti-patterns, such as irregularities in formulas or references to empty cells. Recently, the term "spreadsheet smells" was introduced to indicate some of these errors [24].

The approach of identifying anti-patterns has benefits, such as the fact that the person writing the anti-pattern can use their insight to determine how important a pattern is, or when it should be applied. On the other hand, there are also downsides to creating a catalog of spreadsheet smells based on human experience and effort. First, it takes people's time, identifying a new pattern, implementing it, and testing to see what the false positive and false negative rate is. Second, because this approach often fails to leverage the large volume of existing spreadsheets, such anti-patterns do not necessarily portray an accurate reflection of real errors.

Some of the advances in deep learning served as an inspiration for this work. Deep learning technology has made dramatic strides in the last five years alone [27]. Recurrent neural networks (RNNs) have been applied to speech recognition [23] and image recognition [35]. Deep learning approaches leverage a proven architecture for neural network design with the ability to train the networks with large amounts of data to achieve dramatic results with respect to precision and recall for complex cognitive tasks.

This paper is the first to apply neural networks to the problem of understanding the content and structure of spreadsheets. Given that neural networks are capable of learning complex classification tasks (such determining the breed of a dog in an image), we believe that they can greatly support an individual's understanding of the structure and content of their spreadsheets. Machine learning has been increasingly applied to reasoning about software (e.g., see [13],[28]). Spreadsheets share some things in common with software and also

[1] A note on terminology: following the terminology from Excel, we use the term workbook to describe a file containing multiple worksheets and the term spreadsheet or sheet to refer to the individual sheets in a workbook.



	A	B	C	D	E	F
1		July	August	September	October	November
2	North	4	3	4	2	4
3	East	7	8	9	5	10
4	South	4	5	6	4	7
5	West	5	6	6	5	7
6						
7	Total	20	22	26	16	28

(a) Value view.

	A	B	C	D	E	F
1		July	August	September	October	November
2	North	4	3	4	2	4
3	East	7	8	9	5	10
4	South	4	5	6	4	7
5	West	5	6	6	5	7
6						
7	Total	=SUM(B2:B5)	=SUM(C2:C5)	26	=SUM(E2:E5)	=SUM(F2:F5)

(b) Formula view.

Figure 1: Two views of an example spreadsheet: normal (or value) view and formula view. Note the error in cell D7, which in the value view appears to be the computed sum of column D but in fact is an incorrect number (the sum is 25).

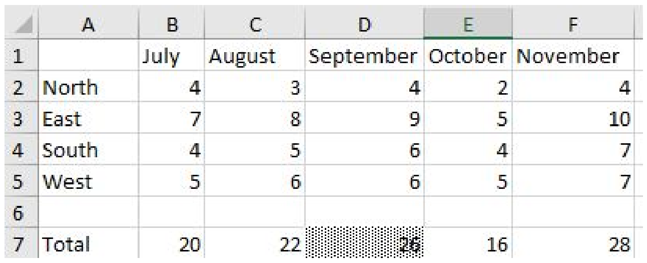
have some unique qualities of their own. In particular, they combine code (such as embedded formulas), data, and graphical layout that other forms of software do not. We believe that the physical layout of spreadsheets, which captures the desire of the author to convey information to the spreadsheet reader, intuitively, leading to visually recognizable patterns, makes then *especially* amenable to using deep learning techniques to reason about.

We use two neural networks including the Feedforward network and the Long Short Term Memory (LSTM) variant of a recurrent neural network (and their parametric variants) as the basis for our classifier. The network's classification task is to predict whether a given cell in a spreadsheet should contain a formula when it currently contains a number.

Motivating example: Figure 1 illustrates such a cell in an example spreadsheet. Figure 1 shows the normal view of the simple spreadsheet, where row 7 appears to contain the total of the cells in the columns above it. Normally such sums are computed using a `SUM` formula, and the user can view all the formulas in the sheet by pressing `Control-`` in Excel.

In the example, we see that while the other columns are computed using `SUM`, column D instead just has a number in row 7, which, while numerically close, does not actually represent the sum of the values in the column. We call such errors "number-where-formula-expected" (NWFE) errors. The job of the classifier is to identify cells like D7 and highlight them to the user. Figure 2 illustrates the output of our tool Melford on this example.

Classifier: To build the classifier, we define an abstraction of the contents of the spreadsheet and use the abstraction over the local spatial neighborhood of a cell to predict its contents. We train the network using a collection of existing spreadsheets and then apply the classifier to new spreadsheets. We compare our approach with a classifier using the same abstraction but using simple statistics instead of deep learning, a classifier using Support Vector Machines (SVM) with the same abstraction, and we also compare against CUSTODES, a



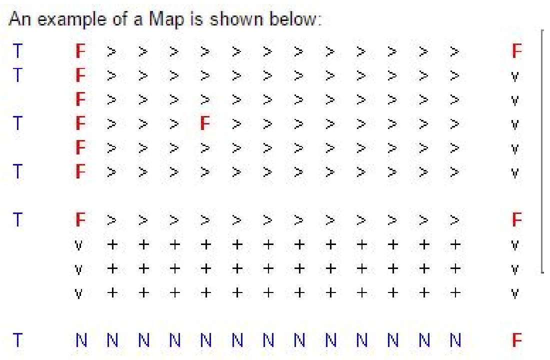
	A	B	C	D	E	F
1		July	August	September	October	November
2	North	4	3	4	2	4
3	East	7	8	9	5	10
4	South	4	5	6	4	7
5	West	5	6	6	5	7
6						
7	Total	20	22	26	16	28

Figure 2: Result of Melford correctly identifying error at D7 in the example spreadsheet.

recent spreadsheet error detection algorithm that uses clustering techniques based on hand-coded features to identify anamolies [14]. In our evaluation, we compare variants of our classifier and CUSTODES using a collection of 268 sheets also used to evaluate CUSTODES. Using the 268 benchmark spreadsheets has the advantage that they have been hand-labeled to identify errors where a number is present where a formula is expected. We show that Melford has a significantly higher precision and recall than our simple statistical classifier and the SVM-based classifier, and that it has an average precision close to CUSTODES with a *significantly lower* false positive rate.

Contributions: The contributions of this paper are as follows.

- This is the first paper to explore the use of multi-layer neural networks to reason about the structure and contents of spreadsheets, specifically for finding errors.
- We show that a simple abstraction of the contents of the spreadsheet, when fed into a neural network, produces a classifier that is effective at finding spreadsheet errors where a formula is expected but a number is present.
- We present measurements of the effectiveness of the approach, both in the training and execution time, and also in the classification effectiveness, applied to spreadsheets used in previous research.



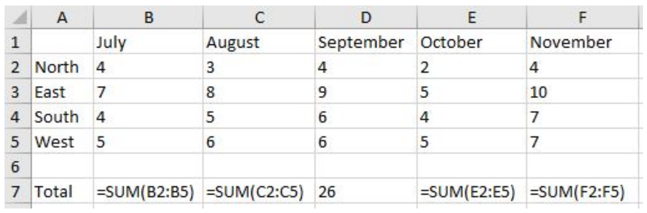
An example of a Map is shown below:

```
T   F > > > > > > > > > > > >   F
T   F > > > > > > > > > > > >   v
    F > > > > > > > > > > > >   v
T   F > > > F > > > > > > > >   v
    F > > > > > > > > > > > >   v
T   F > > > > > > > > > > > >   v

T   F > > > > > > > > > > > >   F
    v + + + + + + + + + + + +   v
    v + + + + + + + + + + + +   v
    v + + + + + + + + + + + +   v

T   N N N N N N N N N N N N N   F
```

Figure 3: Spreadsheet abstraction used in the Spreadsheet Advantage map tool.

- We use the MELFORD system to find true errors that were previously unknown to us in 32 sheets out of 1,513 sheets in the EUSES spreadsheet corpus within a few hours.

Paper organization: The rest of this paper is organized as follows. Section 2 describes the abstractions we define that are used as the input to the neural network training. Section 3 describes the architecture of the network and the parameters used configure it. Section 4 describes our evaluation methods and the benchmarks used. Section 5 contains the results of our evaluation while Section 6 describes related work. We conclude in Section 8.



	A	B	C	D	E	F
1		July	August	September	October	November
2	North	4	3	4	2	4
3	East	7	8	9	5	10
4	South	4	5	6	4	7
5	West	5	6	6	5	7
6						
7	Total	=SUM(B2:B5)	=SUM(C2:C5)	26	=SUM(E2:E5)	=SUM(F2:F5)

(a) Formula view.

```
X X X X X X X X X X
X X X X X X X X X X
X X O S S S S S X X
X X S N N N N N X X
X X S N N N N N X X
X X S N N N N N X X
X X S N N N N N X X
X X O O O O O O X X
X X S F F N F F X X
X X O O O O O O X X
X X X X X X X X X X
X X X X X X X X X X
```

(b) MELFORD abstraction.

Figure 4: Abstraction for spreadsheet in Figure 1.

2. Spreadsheet Abstractions

Our approach is motivated by two observations. First, the human process of finding errors in spreadsheets is at least in part based on visual pattern matching. For example, consider the Map view provided by the commercial spreadsheet debugging tool Spreadsheet Advantage [3] shown in Figure 3. The figure shows how the tool abstracts the cell contents into *classes* of values, and shows the result to the user. It uses 6 symbols: F denotes presence of a formula in a cell, > denotes that the cell has the same formula as the cell on the left, v denotes the same formula as the cell above it, + denotes where both > and v holds, T denotes text, and N denotes a number value. In the example given, the F in the field of ">" symbols stands out as a potential error.

The second observation is that deep learning techniques have proven very successful at learning patterns in very complex data to the degree that in image and speech recognition tasks, they perform close to a human, given enough training data. Using these two observations, we formulate the problem of finding anomalies in spreadsheets as learning a recognition/classification task over an abstraction of a spreadsheet that retains the visual and spatial aspects of the document. The initial abstraction we chose is very simple. We replace each cell with a simple representation of the type of the contents of the cell. The mapping is:

- F – if the cell contains a formula;
- N – if the cell contains a number;
- S – if the cell contains a string;
- O – if the cell is empty;
- X – if the cell is at the boundary of the spreadsheet.
- B – if the cell contains a Boolean value.
- E – if the cell type has an Error value.

Beyond this abstraction, we retain the spatial structure of the cells. For example, the spreadsheet in Figure 1 is represented in Figure 4b. Just as with Spreadsheet Advantage, this view of the spreadsheet provides immediate visual clues about potential anomalies in the contents. Our belief is that we can train neural networks to recognize and identify the error at D7 with high precision and recall.

With this representation, we then pose the recognition of error cells as a classification problem. Given some amount of context around a cell, predict the class of object that the cell contains. Cases where the cell does *not* contain the class that is predicted are flagged as potential errors. Our initial approach to defining context is motivated by image classification approaches used for deep learning.

The classifier's task is to predict the contents of the center cell, given the surrounding cells. In this example, it is not hard to see that based on training the natural prediction would be to predict F, which it in fact does. Given that the actual contents of the cell is an N, this cell is then flagged by MELFORD as a potential error.

Note that this is not the only abstraction we have considered. If our classification required reasoning about the operators within a formula, for instance, to distinguish between sums and products, more complex abstractions would be necessary.

3. Neural Network Model

We now describe the different encodings we apply to the spreadsheet abstractions to generate input vectors and our neural network architecture for learning to identify anomalies from these vectors. For a collection of spreadsheets, we generate a set of training vectors X and the corresponding label set y. For each cell c in a spreadsheet, the training set consists of a vector $X[c]$ that encodes some context around the cell and the label $y[c]$ denotes the desired label dependent on the task. The training vectors are then fed into different neural network architectures to learn the function to label cells.

3.1 Spreadsheet Encodings

Given a spreadsheet abstraction, we need a mechanism to convert them into a set of training vectors. For this work, we present a few encodings for obtaining fixed dimensional encoding vectors since they can be then fed to several neural architectures for learning the labeling functions.

$N \times N$ **window encoder:** The $N \times N$ window encoder creates k fixed dimensional vectors each of size $N^2 - 1$ from a spreadsheet abstraction, where k denotes the number of cells in the spreadsheet. For each cell (i, j) in a spreadsheet T, we generate the following input vector $X[T, i, j]$ and the label $y[T, i, j]$:

$$X[T,i,j] = \{T[k,l] \mid i - N/2 \le k \le i + N/2, k \ne i \\ j - N/2 \le l \le j + N/2, l \ne j\}$$

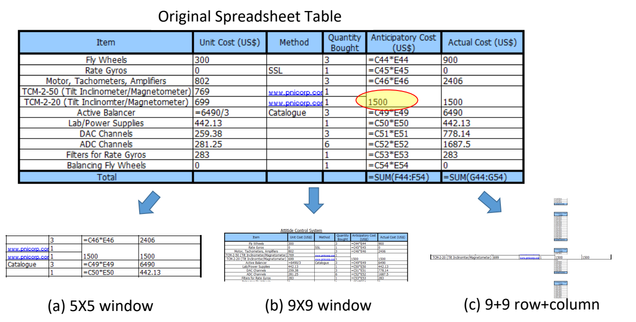
$$y[T,i,j] = \begin{cases} \mathsf{F}, & \text{if } T[i,j] = \mathsf{F} \\ \mathsf{N}, & \text{otherwise} \end{cases}$$

We instantiate the $N \times N$ window encoder for two different values $N = 5$ and $N = 9$. Specifically we consider the cells in a $N \times N$ square around a given cell as context to predict its contents.

An example 5×5 and 9×9 window encoding for a spreadsheet cell is shown in Figure 6(a) and Figure 6(b), respectively. For the cell

```
NNNNN
OOOOO
FF FF
OOOOO
XXXXX
```

Figure 5: MELFORD 5×5 abstract context for cell D7 in Figure 1.



Item	Unit Cost (US$)	Method	Quantity Bought	Anticipatory Cost (US$)	Actual Cost (US$)
Fly Wheels	300		3	=C44*E44	900
Rate Gyros	0	SSL	1	=C45*E45	0
Motor, Tachometers, Amplifiers	802		3	=C46*E46	2406
TCM-2-50 (Tilt Inclinometer/Magnetometer)	769	www.pnicorp.cor	1		
TCM-2-20 (Tilt Inclinomter/Magnetometer)	699	www.pnicorp.cor	1	1500	1500
Active Balancer	=6490/3	Catalogue	3	=C49*E49	6490
Lab/Power Supplies	442.13		1	=C50*E50	442.13
DAC Channels	259.38		3	=C51*E51	778.14
ADC Channels	281.25		6	=C52*E52	1687.5
Filters for Rate Gyros	283		1	=C53*E53	283
Balancing Fly Wheels	0		1	=C54*E54	0
Total				=SUM(F44:F54)	=SUM(G44:G54)

Figure 6: Three fixed dimensional encodings for spreadsheet cells.

D7 in Figure 1, the 5x5 context surrounding it is shown in Figure 5.

Row+column encoder: The $N + N$ row+column encoder creates an input vector of size $2 \cdot (N - 1)$, where the values correspond to the cell abstractions in the row and column of a spreadsheet cell. Specifically, for a cell (i, j) in a spreadsheet table T, we have

$$X[T,i,j] = \{T[k,j] \mid i - N/2 \le k \le i + N/2, k \ne i\} \\ \cup \{T[i,l] \mid j - N/2 \le l \le j + N/2, l \ne j\}$$

$$y[T,i,j] = \begin{cases} \mathsf{F}, & \text{if } T[i,j] = \mathsf{F} \\ \mathsf{N}, & \text{otherwise} \end{cases}$$

We instantiate the row+column encoder for $N = 9$, as shown in Figure 6(c).

3.2 Neural Network Model

Our neural network architecture for the label prediction task is shown in Figure 7. The spreadsheet is first encoded using the encoders described previously to a set of fixed-length vectors. The vectors are then processed by a neural network – a feed-forward network or an LSTM with 128 hidden units in each layer. For feed-forward networks, we use the encoding vector as the input layer, whereas for the LSTM network, we provide the encoding vector as a sequence of abstract symbols (from the abstraction vocabulary). The hidden layer representation is then propagated to a feed-forward layer with n units, where n denotes the vocabulary size.

Finally, we have a softmax layer on top of the final layer to obtain a probability distribution over the set of characters in the vocabulary. Note that for the classification task of predicting whether a cell should be a number or a formula, we only need to learn a binary classifier, where only 1 unit in the final layer would suffice. But, we use a general architecture to also allow for prediction of other abstraction types in the vocabulary for future work in detecting other error types.

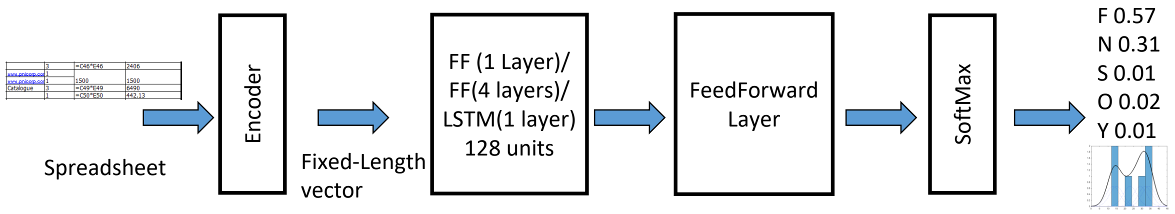


Figure 7: The network architecture for learning the labels for spreadsheet encodings.

The feed-forward networks consist of layers of computational units, where each layer consists of a set of artificial neurons, which use a non-linear function to be activated (1) or not (0). The layers are connected with weighted edges, which allows feed-forward networks to learn non-linear functions. We evaluate the feed-forward network with different numbers of hidden layers. The LSTM network model is a variant of the recurrent neural network (RNN) model, which allows for encoding variable-length vectors as sequences and learning longer contexts in sequences. LSTMs have previously been shown to outperform other alternative recurrent models for the task of handwriting recognition and speech recognition. The network is trained end-to-end over millions of spreadsheet vectors obtained from a large number of training spreadsheets.

4. Experimental Methods

In this section, we discuss the methods and our training setup used to evaluate the performance of Melford.

4.1 Training Data

For training our neural network models, we used workbooks from the VEnron corpus available from `http://sccpu2.cse.ust.hk/venron/` and described in a recent paper [19]. The corpus contains 7,296 workbooks that were made public in the litigation surrounding the Enron Corporation that occurred in the early 2000's.

The sheets are from a diverse set of sources including financial spreadsheets, and the related research clustered them into 360 distinct clusters, each of which represents the evolution of each sheet over time. We trained our network models using 13,377 sheets obtained from these workbooks.

We also trained our model using 17,719 sheets from a private collection of diverse workbooks available to us (we call these "ExcelSamples"). This collection contains 9,512 Excel workbooks. Because the workbooks do not represent multiple instances of the same workbook through its history, we believe that our collection is in fact *more* diverse than the VEnron collection. While we are not able to share these workbooks, we plan to share the models learnt on this dataset.

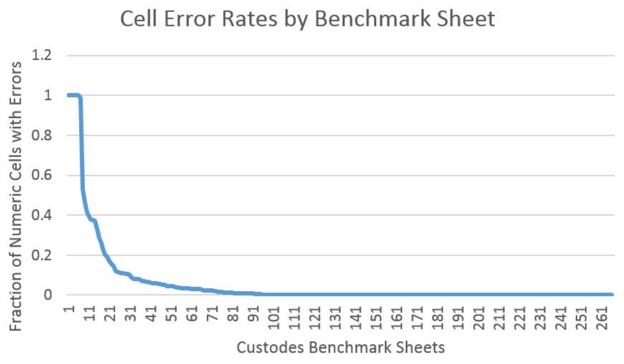


Figure 8: Custodes benchmark sheets sorted by the fraction of numeric cells considered to be errors.

4.2 Evaluation Benchmarks

For the purpose of evaluating our models, we use the Custodes benchmark suite available from the Custodes project at `http://sccpu2.cse.ust.hk/custodes/`. The suite contains a collection of 70 Excel workbooks selected from the EUSES spreadsheet corpus [22] containing 268[2] different sheets. The Custodes project hand-labeled the cells in the sheets to provide ground-truth with respect to several different kinds of errors including errors where a formula is expected but a number is present.

Because Melford is trained to specifically detect number-where-formula-expected (NWFE) errors, we compare the performance of different algorithms on this subset of errors in the Custodes benchmark workbooks. Fortunately there are many such errors in these workbooks — our count for total NWFE errors across the 268 sheets is 1,707. Because this number is so high, we investigated the distribution of hand-labeled NWFE errors in the Custodes workbooks and found that there are several cases where a large fraction of the numbers in the worksheet are considered errors by the Custodes labeling.

[2]While the `groundtruth` directory from Custodes contains 290 sheets, 22 of those sheets have no cells containing numbers, and are thus omitted from our analysis which only makes predictions on cells containing numbers.

In Figure 8, we show the fraction of cells containing numbers marked as true errors by the CUSTODES project. The figure shows that in a small number of sheets, almost every cell containing a number is considered an error. For example, in the workbook `inter2.xls` in the sheet "Summary of Vital Statistics" out of 2,212 cells, 422 are considered errors by the CUSTODES labeling.

Another set of 7 sheets from a single workbook each have either 77 or 78 errors with 100% of all the cells containing numbers labeled as errors. Our belief is that cases where a large fraction of all cells in a sheet contain errors is unlikely in practice, because human auditors can easily catch such errors. As a result, we focus our evaluation on a subset of the CUSTODES spreadsheets, in which the total fraction of cells containing errors is less than 10% (the low-error subset). This filter removes approximately 10% of the CUSTODES spreadsheets from the benchmark suite. The number of total cells with true errors is reduced to 474 out of 42,269 cells containing numbers. In Section 5, we present results for both the low-error subset of CUSTODES and the full CUSTODES benchmark suite.

In addition the CUSTODES dataset, we also evaluated our learnt model on another test dataset obtained from the EUSES collection of 720 financial spreadsheets. This dataset is much larger than CUSTODES, but these spreadsheets do not have the ground truth labeling. We wanted to observe the experience of an auditor using MELFORD on a new dataset to find the NWFE errors.

4.3 Baseline Comparison

In addition to comparing MELFORD to CUSTODES, we also compare it against two simpler baselines.

Statistical Classifier: The first baseline is a simple statistical classifier (which we call "Stencil") based on remembering every 5×5 context and predicting based on the frequency of outcomes in the training set. For example, if the 5×5 stencil shown in Figure 5 occurred 10 times in the training set and the center cell contained an F 9 times and an N once, we record this exact distribution ($Num_F = 9$) and ($Num_N = 1$).

As with our network models, we use this model as a classifier to detect instances where F is predicted but N is present. We set a threshold for prediction based on the ratio of observations where F was observed compared to how many times either F or N was observed. If $Num_F/(Num_F + Num_N) > 0.5$), we flag the cell as an error.

For training, we used the ExcelSamples training set described above. In that collection of spreadsheets we observed and recorded 3,553,482 different contexts. Since our baseline approach can only predict specific contexts that it has seen, a comparison of MELFORD with the baseline gives us an understanding of how effective the learning model is at generalizing from the raw data.

SVM Classifier: The second baseline we use is a classifier based on Support Vector Machines (SVM). The spreadsheets are encoded to a fixed dimension 5×5 context vector for both the training sets and the vectors are then used to train the SVM model. We use the default SVM model from `scikit` that uses the `rbf` kernel and also trained it with the contexts observed in the ExcelSamples dataset. For a fair comparison with our network models, we do not provide any additional features to the encoding vector other than the abstraction of the cells in the 5×5 context.

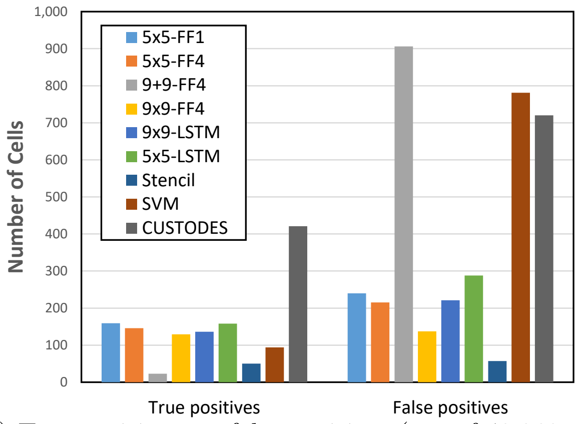
5. Evaluation

In this section, we evaluate the effectiveness of MELFORD in finding number-where-formula-expected (NWFE) errors in spreadsheets. Our goal is to answer the following questions:

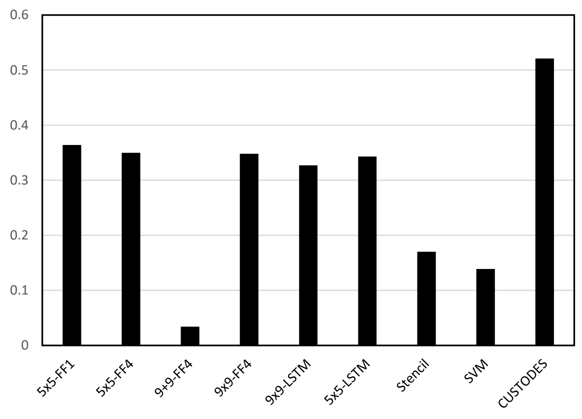
- **RQ1**: Does the MELFORD approach find real errors, and what is the false positive and false negative rate of the method?
- **RQ2**: How does the deep learning approach compare to previous techniques, including CUSTODES, SVM (support vector machine), and a simple statistical baseline?
- **RQ3**: What is the performance of the deep learning method, both in the time it takes to *train* the model and the *execution* or application time, when using the model to predict errors?
- **RQ4**: How sensitive is the approach to model parameters such as the contents of the training corpus and the threshold used to decide whether to predict an error?

5.1 Training Times

For training the network models, we obtain around 9.1 million vectors from the ExcelSamples dataset. We divide the vectors into 80% for training and the remaining 20% for validation. We train the networks for 50 *epochs*, where each epoch completes a forward and backward pass in the network for all the training vectors. We use a batch size of 128, i.e. each forward/backward pass consists of 128 training examples. The models took from 18 hours to 76 hours to train, depending on the configuration. In some training runs, specifically for the larger 9×9 models, the machine ran out of memory before finishing 50 epochs. For such cases, we use the last saved epoch model for evaluation. The models were trained on a machine with a 2.8GHz Intel Xeon CPU with 64 GBs of RAM and a 12GB Nvidia Tesla K40m GPU. We used the keras deep learning li-



(a) True positives vs. false positives (out of 42,269 cells.)



(b) F1 score values for the different models.

Figure 9: Classification effectiveness of the different models for the low-error subset of CUSTODES benchmarks

brary [15] to implement our network models, which underneath use the Theano deep learning framework [33].

5.2 Classification Effectiveness

The number of true and false positives reported by the various classifiers we consider and their corresponding F1 score is shown in Figure 9. All neural network classifiers and the baseline classifiers (`SVM` and `Stencil`) were trained on the ExcelSamples data. Each layer in the neural network model (feed-forward or LSTM) consists of 128 hidden units. Specifically,

- `Stencil` denotes our baseline classifier consisting of 5x5 encoding vectors;
- `SVM` denotes the SVM classifier trained with 5x5 encoding vectors;
- `CUSTODES` shows the result of running the CUSTODES tool;
- `5x5-FF1` denotes a model with 1 feed-forward layer trained on 5x5 encoding vectors;
- `5x5-FF4` denotes a model with 4 feed-forward layers trained on 5x5 encoding vectors;
- `9+9-FF4` denotes a model with 4 feed-forward layers trained on 9+9 encoding vectors;
- `9x9-FF4` denotes a model with 4 feed-forward layers trained on 9x9 encoding vectors;
- `9x9-LSTM` denotes a model with 1 LSTM layer trained on 9x9 encoding vectors;
- `5x5-LSTM` denotes a model with 1 LSTM layer trained on 5x5 encoding vectors.

True and false positives: In the low-error subset of the CUSTODES benchmark workbooks, there were 42,269 cells that contained numbers and 474 of those were labeled by hand as true NWFE errors. Figure 9 shows the absolute number of true and false positives reported by the various classifiers for the low-error subset of CUSTODES spreadsheets. We observe that CUSTODES finds the most true positives (421/474) but also reports a large number of false positives (720). We believe this result happens because CUSTODES is quite aggressive at growing clusters of related cells, to the point that in some cases it over-generalizes, resulting in more false positives.

All network models with $N \times N$ encodings significantly outperform the `Stencil` and `SVM` baseline models. The `Stencil` baseline predicts only 50 true positives and 57 false positives. The `SVM` baseline predicts 94 true positives, but 781 false positives. The performance of different models with $N \times N$ encodings is comparable. The best model amongst them is `5x5-FF1` which finds 159 true positives with 240 false positives. In general, adding additional feed-forward layers do not seem to improve the model performance as the best `5x5-FF4` model finds 146 true positives with 215 false positives. The larger window size also does not add to the model performance as the best `9x9-FF4` model finds 129 true positives with 137 false positives. Using an LSTM layer instead of a feed-forward layer seems to slightly degrade the performance. Finally, the `9+9-FF4` model performs the worst as it finds a large number of false positives. This result shows the importance of having all the cells in the neighborhood window, as opposed to only the row and column cells.

The F1 score for each model is shown in Figure 9(b). The precision and recall values are computed by accumulating all the test spreadsheet cells together in one large list of cells. As described earlier, `5x5-FF1` models seems to perform the best amongst all network models with an F1 score of 0.365. Other $N \times N$ models have F1 scores in the range of 0.32– 0.35, which are all significantly higher than the baseline scores of 0.17 (`Stencil` and 0.14 (`SVM`). The CUSTODES system achieves an F1 score of 0.521 on this dataset.

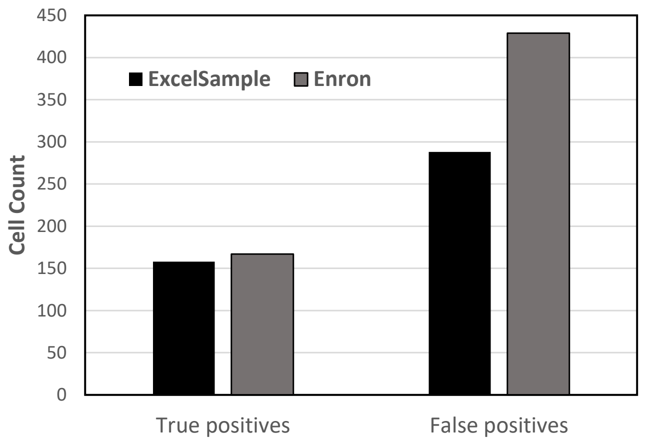


Figure 10: The effect of different training data on the performance of `5x5-LSTM` network model.

In conclusion, the 5x5 context trained with the ExcelSamples dataset performed the best amongst the Melford configurations, detecting significantly fewer true errors than Custodes but also reporting significantly fewer false positives. Melford predicts that less than 1% (0.68%) of cells contain errors and is correct about 40% of the time for the Custodes benchmarks.

5.3 Effect of Training Data

We evaluate the effect of different training data on the performance of network models by training the `5x5-LSTM` model on both the ExcelSamples dataset and the Enron dataset. The performance of the two learned models is shown in Figure 10. The model learned using the ExcelSample dataset performs much better than the model learned with the Enron dataset. Our hypothesis for this performance is that the ExcelSamples dataset consists of more diverse set of spreadsheets as opposed to the Enron dataset, which is likely also more representative of the Custodes evaluation dataset.

5.4 Effect of Threshold

The learned network models generate a probability distribution over the alphabets in the encoding vocabulary. In our case, since we are only considering the label of cells as `N` or `F`, the learned model predicts each one with some probability p. We use a threshold value t such that when $p > t$, we report the cell as erroneous. In this experiment, we vary the value of threshold to observe its effect on performance of the `5x5-FF1` model as shown in Figure 11. We can observe that as we increase the threshold value, the number of true positives and false positives both decrease. For threshold value of 0.9, we can observe that the `5x5-FF1` model makes more than 50% correct error cell predictions. For this model, the best F1 score is obtained for the threshold value of 0.5(0.365).

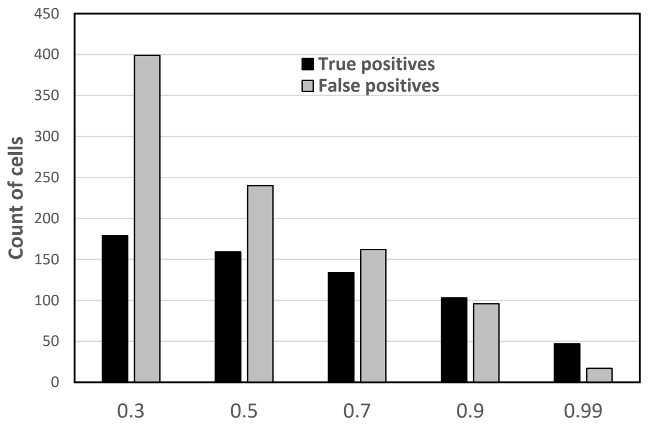


Figure 11: The effect of varying threshold value on the performance of the `5x5-FF1` model. True and false positive counts induced by different thresholds values are shown on the x axis.

5.5 Model Execution Time

The evaluation time for running Melford on the Custodes benchmarks for different network models is shown in Figure 12. As can be observed the execution times for feed-forward networks is about 1,000 seconds for 268 benchmark sheets, whereas the LSTM networks take much longer time (about 4 to 7 times). This is because the LSTM architecture takes one element of a sequence at a time and unrolls the computations. The feed-forward models take on average about 0.025 seconds per cell, whereas the LSTM models take on average about 0.12 seconds per cell.

5.6 Full Custodes Results

We now present the results of our learnt models on the full Custodes dataset containing even spreadsheets where more than 10% of the cells are labeled as NWFE

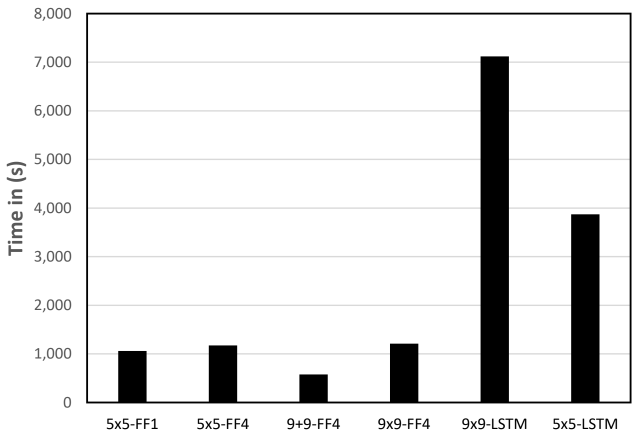


Figure 12: The total execution time for different learned models on 268 Custodes spreadsheets.

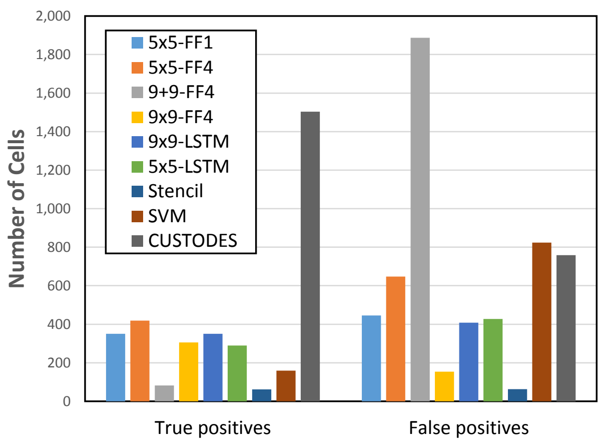


Figure 13: True positives vs false positives for different models on the full CUSTODES dataset of 268 spreadsheets.

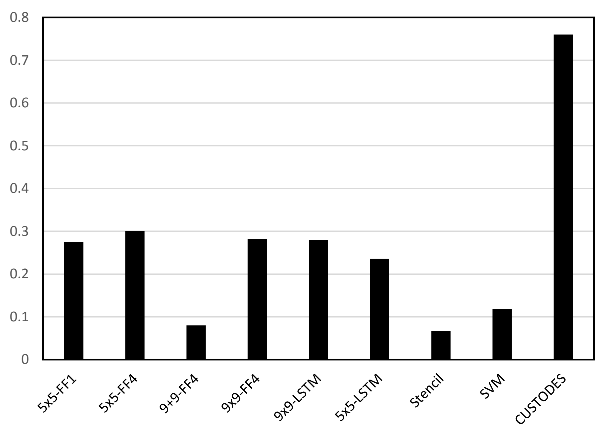


Figure 14: The F1 scores for different models on the full CUSTODES dataset of 268 spreadsheets.

errors. The results for true and false positives count for different models are shown in Figure 13. For the complete dataset, CUSTODES reports 1503 true positives and 759 false positives. Note that in the spreadsheets containing a higher fraction of errors, the CUSTODES approach does exceedingly well, detecting almost all the true errors and adding few additional false positives compared with the low-error subset.

The MELFORD model that finds the most true positives on this dataset is `5x5-FF4`. It finds 419 true positives and 647 false positives. Surprisingly, the `9x9-FF4` MELFORD model finds significantly more (305) true positives and few additional (154) false positives, compared to the performance on the low-error subset, giving it a success rate of 66.5% and the best ratio of true to false positives of any MELFORD model. For the full CUSTODES benchmarks, the `9+9-FF4` model, the `SVM` classifier, and the `Stencil` classifier all have poor performance. The F1 score for each network model is shown in Figure 14. The CUSTODES model has an F1 score of 0.76, whereas the best network model `5x5-FF4` has an F1 score of 0.3.

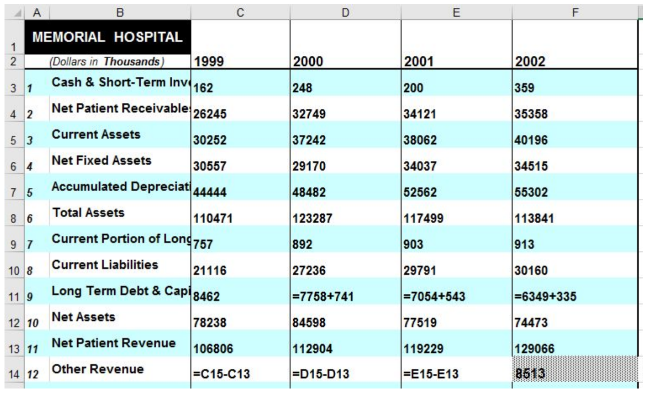
5.7 Direct Experience with MELFORD

In this section, we discuss our experience applying MELFORD to analyze spreadsheets not in the CUSTODES benchmark suite. We chose the entire EUSES collection of 720 financial spreadsheets as our test (a collection much larger than the 70 workbooks in the CUSTODES benchmarks). Our goal was to demonstrate that MELFORD was effective at finding real bugs in spreadsheets that we had not previously looked at. Of the original 720 financial workbooks, we removed 24 because our tool was unable to parse the character sets present in them, leaving 696 workbooks containing a total of 1,513 individual sheets.

We used the `5x5-FF1` model to process all the workbooks in the EUSES financial spreadsheet collection. The total time to analyze the 696 workbooks was just under 3 hours of wallclock time on a workstation class desktop machine (see more details about the performance below). Our tool reported 818 out of a total of 218,438 numeric cells (an error reporting rate of 0.037%) as potential errors in 227 of the 1,513 sheets. We followed up by manually inspecting those sheets to determine if the results were true positives or not. Over the course of 3 hours, one of us inspected each sheet and determined whether we believed the highlighted cell was indeed an error. In many cases we were able to confirm that a cell contained an error because when we replaced the number with the use of an adjacent formula, the formula correctly computed the number that was replaced, indicating that the author had inadvertently replaced a formula with a number.

Based on our analysis, we identified true positive errors, all previously unreported, in 32 sheets from 26 workbooks. We consider the ability to quickly find so many true errors with no prior familiarity with the spreadsheets an accurate test of how effective our tool will be in practice when applied to real spreadsheets.

To illustrate how we used MELFORD in action, Figure 15 shows the formula view of the output of MELFORD for the MEMORIAL sheet of the `hospital-dataset2002.xls` workbook. The figure shows that the adjacent cells contain formulas whereas cell F14 contains a number. Replacing the number in F14 with the formula in E14 produces the same number as a result. The entire list of true positive errors and MELFORD-annotated spreadsheets accompany this paper as a zipped directory of Supplementary Material.



	A	B	C	D	E	F
1		MEMORIAL HOSPITAL				
2		(Dollars in Thousands)	1999	2000	2001	2002
3	1	Cash & Short-Term Inv	162	248	200	359
4	2	Net Patient Receivable	26245	32749	34121	35358
5	3	Current Assets	30252	37242	38062	40196
6	4	Net Fixed Assets	30557	29170	34037	34515
7	5	Accumulated Depreciati	44444	48482	52562	55302
8	6	Total Assets	110471	123287	117499	113841
9	7	Current Portion of Long	757	892	903	913
10	8	Current Liabilities	21116	27236	29791	30160
11	9	Long Term Debt & Capi	8462	=7758+741	=7054+543	=6349+335
12	10	Net Assets	78238	84598	77519	74473
13	11	Net Patient Revenue	106806	112904	119229	129066
14	12	Other Revenue	=C15-C13	=D15-D13	=E15-E13	8513

Figure 15: Example of Melford finding a true error (cell F14) in the MEMORIAL sheet of the workbook `hospital-dataset2002.xls` in the EUSES financial spreadsheet collection.

5.8 Classification Execution Time

Our performance results were obtained using an HP Z820 Workstation with 64 gigabytes of memory running Windows 8.1 Enterprise. We measured the execution time for Melford to classify all the numeric cells for the entire set of 696 EUSES financial spreadsheets and plotted the total classification time against the number of numeric cells in each sheet. The result, shown in Figure 16 shows that the total time scales linearly with the number of cells. A linear regression fits the data with an intercept of 3.56 seconds and a per cell cost of 0.011 seconds.

This measurement reflects that our current C# implementation of Melford starts up a Python virtual machine to process all the cells in a sheet to call the keras library and so we believe that the 3.5 second constant reflects that overhead. A faster implementation that evaluates the model in C# itself without having to start a separate Python process would significantly decrease the overhead.

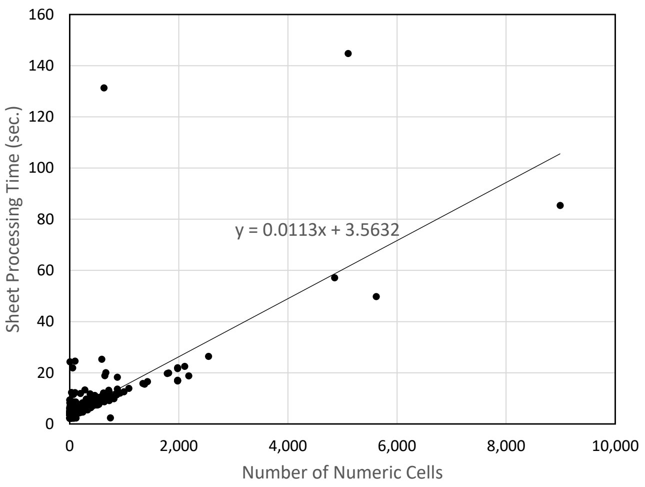
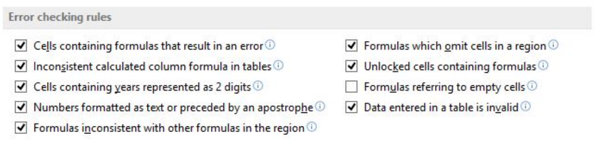


Figure 16: Classification time (y-axis, in seconds) vs. the number of cells classified (x-axis).



Error checking rules

- ✓ Cells containing formulas that result in an error
- ✓ Inconsistent calculated column formula in tables
- ✓ Cells containing years represented as 2 digits
- ✓ Numbers formatted as text or preceded by an apostrophe
- ✓ Formulas inconsistent with other formulas in the region
- ✓ Formulas which omit cells in a region
- ✓ Unlocked cells containing formulas
- ☐ Formulas referring to empty cells
- ✓ Data entered in a table is invalid

Figure 17: Error checking rules in Excel 2013.

6. Related Work

In this section, we consider the most closely related work on finding errors in spreadsheets as well as other work that has applied neural networks for reasoning about software. Because the issue of correctness of spreadsheets has been of importance since they were first introduced, there have been a number of approaches to finding and fixing errors. A more comprehensive discussion of approaches to finding and fixing errors in spreadsheets was recently published by Jannach et al. [26].

6.1 Excel Built-in Support

Microsoft Excel [2] includes built-in support to help users identify errors in spreadsheets. Figure 17 shows a dialog with a list of errors that Excel 2013 highlights. The approach supported by Excel is based on recognizing common pre-defined patterns of errors and flagging specific cells that the patterns identify as potentially wrong. Some of the rules check the contents of cells (e.g., checking that the result of a computation is not itself an error). Other checking involves the relation between cells such as if the calculations of formulas over columns use inconsistent ranges or formulas omit cells in a contiguous region. The Excel approach relies on a fixed, manually curated set of rules that check for cases that are common and likely to be true errors.

6.2 Spreadsheet Smells

In software development, the approach of using automation to detect patterns of coding that are associated with bad design or bad practices is called detecting "code smells" [34]. Recently, this approach has also been applied to detecting errors in spreadsheets [5, 8, 16, 17, 24]. The key elements of this approach have similarities to the approach implemented in Microsoft Excel — common patterns of bad coding or bad design practices are cataloged and instances of occurrences of such practices are highlighted for inspection. This work has evolved in the last few years and the catalog of the kinds of errors that such approaches detect has expanded.

For example, in recent work [5], the smells include: cell value deviation (either numeric or via string distance), empty cells in unexpected places, cells that don't have the same type as other cells in a column/row, bad references (to empty cells), computations with many references or resulting from a long chain of dependences, and formulas with high conditional complexity (inspired directly from code smells in programming).

6.3 Custodes

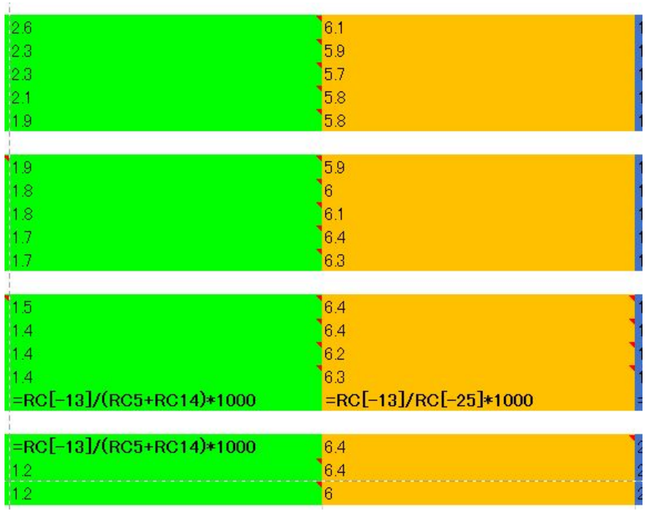
More recent work has focused on combining spreadsheet smells with looking for cells that are classified as outliers [14, 18]. The work closest to ours is Custodes [14], which uses clustering techniques to identify groups of cells believed to have common properties and then looking for anti-patterns to find outliers in these groups. Specifically, Custodes uses a two-stage clustering strategy to define groups of cells believed to share a common role.

The first stage clusters cells that contain similar ASTs and have similar dependences while the second stage adds cells to the cluster based on a set of "weak" features that include cell address, labels, etc. Once the clusters are formed they look for specific kinds of anomalies that represent potential errors such as a missing formula or a hard-coded constant (which is equivalent to our number-where-formula-expected error).

Custodes differs from Melford in the basic approach used. Our approach requires no specific clustering algorithm or enumeration of strong and weak clustering features but reasons about the contents of cells based on local patterns that we learn automatically through training. Custodes is aggressive at creating clusters and classifying cells as errors, allowing it to generalize from a small number of cells to predict that an entire column is erroneous. Because it uses more local reasoning, Melford will not make such aggressive generalizations, resulting in both a lower true positive and false positive rate as our results demonstrate.

Figure 18 illustrates some of the pros and cons of the Custodes approach. The figure presents the formula view of the sheet, with formulas only appearing in two of the rows. Each cell marked with a red triangle indicates a true error. The cells filled in green and yellow indicate clusters that the Custodes algorithm infers. Note that a large number of cells in each column are included in clusters despite the presence of only one or two formulas which are used as a basis for forming the clusters.

In the case of the green cluster, generalizing from a small number of formulas allows Custodes to correctly identify numerous true positive errors, whereas we see with the yellow cluster that many of the cells in the cluster are not errors (indicated by the absence of a red triangle in the upper right cells). Overall, in this sheet,



2.6	6.1
2.3	5.9
2.3	5.7
2.1	5.8
1.9	5.8
1.9	5.9
1.8	6
1.8	6.1
1.7	6.4
1.7	6.3
1.5	6.4
1.4	6.4
1.4	6.2
1.4	6.3
=RC[-13]/(RC5+RC14)*1000	=RC[-13]/RC[-25]*1000
=RC[-13]/(RC5+RC14)*1000	6.4
1.2	6.4
1.2	6

Figure 18: Custodes labeling for the workbook `inter2.xls` in the sheet "Summary of Vital Statistics."

out of 422 true errors, Custodes correctly finds 387 and incorrectly reports 21.

6.4 Other Approaches

Another approach to finding bugs in spreadsheets involves inferring types and units, and reasoning about their compatibility [4, 6, 20]. Spreadsheets typically have a limited type system involving types such as numbers, strings, formulas, etc. These approaches attempt to define collections of cells with a common type by examining values (e.g, strings that contain numeric values) and the spreadsheet layout and using them as clues for inference. While prior work has attempted to infer the spatial structure of tables, including identifying headers [4], no prior work has used neural networks for this purpose.

As mentioned, finding numeric outliers in spreadsheets is one of the "smells" considered in prior work. Auditing tools that find values that fall outside of expected distributions can be effective. For example, techniques to determine if the values in spreadsheets follow Benford's Law [1] are used in forensic analytics to detect spreadsheets with fraudulent data in them [12, 30].

More recent work goes beyond examining individual values and considers the impact that values have on chains of *computation*, including the "outputs" of the spreadsheet [10]. This work directs the attention of the spreadsheet author not to erroneous values, but instead to those values that have the most impact on the conclusions drawn from the spreadsheet.

6.5 Neural Networks for Software Reasoning

There has been a lot of recent interest in using neural networks for analysis of programs. Neural networks have recently been used to learn language model embedding of introductory programming code, to automatically cluster and propagate teacher feedback [32], and for fixing syntax errors in student assignments [11]. A convolution network with attention mechanism was also recently applied to perform summarization of source code snippets into descriptive function name-like summaries [7].

Mou et al. [29] propose a novel tree-based convolutional neural network (TBCNN) for encoding programs, and they use the learnt encodings to classify short source code snippets obtained from a programming contest according to their functionality. Our work is inspired by the use of neural networks to the domain of software reasoning. Although spreadsheets can be thought of as a form of software, they are quite different from general programs in the sense that there is a visual layout aspect to it (similar to pictures), unlike general-purpose programs.

Neural networks are also used in detecting malware. Firdausie et al. provide a survey of runtime behavioral techniques for malware detection [21]. Other projects have applied neural networks to static malware classificaion tasks as well [36, 37].

7. Threats To Validity

A threat to internal validity is that we assume that the spreadsheet datasets we use for training our models do not have any anomalies. We try to minimize this threat by using two quality training datasets — VEnron and ExcelSamples, which come from reputable sources as opposed to random spreadsheets on the web. Another internal threat for MELFORD's evaluation results on the CUSTODES benchmark set is that the ground truth was obtained with manual labeling and it might have some incorrect labels as a result. In fact, we have found a small number of such labeleling mistakes during a manual analysis. Another internal threat of using the CUSTODES dataset is the representativeness of benchmark set for real-world spreadsheets. To mitigate these issues, we evaluated our models on the EUSES financial spreadsheets dataset. Since these spreadsheets do not have ground truth associated with it, we had to perform a manual analysis of reported issues, which is another threat to internal validity.

An external threat to validity is that the ExcelSamples data can not be shared publicly. For mitigating this threat, we are planning to at least make our learned models public, so that at least comparisons can be performed against these models.

8. Conclusion

This paper proposes MELFORD, a tool for finding errors in spreadsheets. The specific errors we detect are cases where an author has placed a number in a cell where there should be a formula, such as in the row totaling the numbers in a column. We use a spatial abstraction of the cells around a particular cell to build a classifier that predicts whether a cell should contain a formula whenever it contains a number.

MELFORD uses a simple spreadsheet abstraction, requires no labeled data, and allows us to rapidly explore potential new classifiers to improve the effectiveness of the technique. Our classifier has a low false-positive rate and finds more than 100 real errors in a collection of 70 benchmark workbooks. We also applied MELFORD to almost all of the financial spreadsheets in the EUSES corpus and within hours confirmed real errors that were previously unknown to us in 26 of the 696 workbooks.

We have shown that with neural networks, a simple abstraction of a spreadsheet combined with a moderate corpus of less than 20,000 files can be used to create a classifier that is both fast and effective. We anticipate that we can extend this work in two important ways.

- First, we believe that there are many additional ways in which the contents of a spreadsheet can be abstracted and that this additional information can be used to build better classifiers. Specifically, we currently do not encode any information about the actual formulas, what cells they reference, etc. Likewise, we do not use any information about the presentation of the spreadsheet, such as font information, color, etc., all of which are often used by humans as a way to convey information to viewers. Finally, we believe that incorporating the contents of strings (such as the use the word "Total") and numbers can improve the accuracy of the classifier.
- Another direction we plan to take this work is to consider other classification tasks. For example, recently it was discovered that many spreadsheets containing genome data had cells containing errors, where genome sequences had been accidentally coerced to dates and numbers [25]. We can train the classifier to detect such cells and highlight them. Other kinds of anomalies we believe can be detected with our approach include incorrect cell ranges in formulas and incorrect operators. Beyond finding errors, we believe it should be possible to extract table structure (identifying the headers, boundaries, etc.) from spreadsheets.

We believe that neural networks have a great deal of potential for helping users reason quickly and effectively about the contents of a spreadsheet. We hope that this paper inspires other researchers to consider the exciting intersection of deep learning methods and spreadsheet correctness.

References

[1] Benford's Law Wikipedia page. https://en.wikipedia.org/wiki/Benford

[2] Microsoft Excel Wikipedia page. https://en.wikipedia.org/wiki/Microsoft_Excel.

[3] Spreadsheet Advantage excel add-in. http://www.spreadsheetadvantage.com/features_map.php.

[4] R. Abraham and M. Erwig. Header and unit inference for spreadsheets through spatial analyses. In *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*, pages 165–172. IEEE, 2004.

[5] R. Abreu, J. Cunha, J. P. Fernandes, P. Martins, A. Perez, and J. Saraiva. Smelling faults in spreadsheets. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pages 111–120. IEEE, 2014.

[6] Y. Ahmad, T. Antoniu, S. Goldwater, and S. Krishnamurthi. A type system for statically detecting spreadsheet errors. In *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on*, pages 174–183. IEEE, 2003.

[7] M. Allamanis, H. Peng, and C. A. Sutton. A convolutional attention network for extreme summarization of source code. In *ICML*, pages 2091–2100, 2016.

[8] S. Badame and D. Dig. Refactoring meets spreadsheet formulas. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 399–409. IEEE, 2012.

[9] L. B. Baker. Solarcity advisor lazard made mistake in tesla deal analysis. http://www.reuters.com/article/us-solarcity-lazard-idUSKCN11635K.

[10] D. W. Barowy, D. Gochev, and E. D. Berger. Checkcell: data debugging for spreadsheets. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, pages 507–523. ACM, 2014.

[11] S. Bhatia and R. Singh. Automated correction for syntax errors in programming assignments using recurrent neural networks. *CoRR*, abs/1603.06129, 2016.

[12] R. J. Bolton and D. J. Hand. Statistical fraud detection: A review. *Statistical science*, pages 235–249, 2002.

[13] Y. Brun and M. D. Ernst. Finding latent code errors via machine learning over program executions. In *Proceedings of the 26th International Conference on Software Engineering*, pages 480–490. IEEE Computer Society, 2004.

[14] S.-C. Cheung, W. Chen, Y. Liu, and C. Xu. Custodes: automatic spreadsheet cell clustering and smell detection using strong and weak features. In *Proceedings of the 38th International Conference on Software Engineering*, pages 464–475. ACM, 2016.

[15] F. Chollet. keras. https://github.com/fchollet/keras, 2015.

[16] J. Cunha, J. P. Fernandes, P. Martins, J. Mendes, and J. Saraiva. Smellsheet detective: A tool for detecting bad smells in spreadsheets. In *Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on*, pages 243–244. IEEE, 2012.

[17] J. Cunha, J. P. Fernandes, H. Ribeiro, and J. Saraiva. Towards a catalog of spreadsheet smells. In *Computational Science and Its Applications–ICCSA 2012*, pages 202–216. Springer, 2012.

[18] W. Dou, S.-C. Cheung, and J. Wei. Is spreadsheet ambiguity harmful? detecting and repairing spreadsheet smells due to ambiguous computation. In *Proceedings of the 36th International Conference on Software Engineering*, pages 848–858. ACM, 2014.

[19] W. Dou, L. Xu, S.-C. Cheung, C. Gao, J. Wei, and T. Huang. Venron: a versioned spreadsheet corpus and related evolution analysis. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 162–171. ACM, 2016.

[20] M. Erwig and M. Burnett. Adding apples and oranges. In *Practical Aspects of Declarative Languages*, pages 173–191. Springer, 2002.

[21] I. Firdausi, A. Erwin, A. S. Nugroho, et al. Analysis of machine learning techniques used in behavior-based malware detection. In *Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on*, pages 201–203. IEEE, 2010.

[22] M. Fisher and G. Rothermel. The euses spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–5. ACM, 2005.

[23] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[24] F. Hermans, M. Pinzger, and A. van Deursen. Detecting code smells in spreadsheet formulas. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 409–418. IEEE, 2012.

[25] C. Ingraham. An alarming number of scientific papers contain Excel errors. https://www.washingtonpost.com/news/wonk/wp/2016/08/26/an-alarming-number-of-scientific-papers-contain-excel-errors/, 2016.

[26] D. Jannach, T. Schmitz, B. Hofer, and F. Wotawa. Avoiding, finding and fixing spreadsheet errors–a survey of automated approaches for spreadsheet qa. *Journal of Systems and Software*, 94:129–150, 2014.

[27] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[28] R. Malhotra. A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing*, 27:504–518, 2015.

[29] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin. Convolutional neural networks over tree structures for programming language processing, 2016.

[30] M. Nigrini. *Benford's Law: Applications for forensic accounting, auditing, and fraud detection*, volume 586. John Wiley & Sons, 2012.

[31] R. R. Panko. Spreadsheet errors: What we know. what we think we can do. *arXiv preprint arXiv:0802.3457*, 2008.

[32] C. Piech, J. Huang, A. Nguyen, M. Phulsuksombati, M. Sahami, and L. J. Guibas. Learning program embeddings to propagate feedback on student code. In *ICML*, pages 1093–1102, 2015.

[33] T. T. D. Team, R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

[34] E. Van Emden and L. Moonen. Java quality assurance by detecting code smells. In *Reverse Engineering, 2002. Proceedings. Ninth Working Conference on*, pages 97–106. IEEE, 2002.

[35] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2015.

[36] W. Yu, L. Ge, G. Xu, and X. Fu. Towards neural network based malware detection on android mobile devices. In *Cybersecurity Systems for Human Cognition Augmentation*, pages 99–117. Springer, 2014.

[37] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue. Droid-sec: Deep learning in android malware detection. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 371–372, 2014.