EqnMaster: Evaluating Mathematical Expressions with Generative Recurrent Networks

Amani V. Peddada
Department of Computer Science
Stanford University
Stanford, CA 94305
amanivp@stanford.edu

Arthur L. Tsang
Department of Computer Science
Stanford University
Stanford, CA 94305
atsang2@stanford.edu

Abstract

In this project, we seek to use computational models to infer mathematical structure from purely symbolic inputs. Though there are existing systems for parsing mathematical expressions, there have been only limited approaches to apply *learning-based* algorithms to this inherently subtle task. This project therefore proposes an integrated methodology that applies various configurations of neural networks to analyze sequences of mathematical language. We define both a discriminative task – in which we pursue the *verification* of a given equation – and a generative task – where we predict the *evaluation* of an input mathematical expression. We test our models on a novel dataset, and gain insight into the workings of our models by learning underlying representations of our data.

1 Introduction

The language of arithmetic is an inherently rich yet remarkably unexplored subject. It remains a topic unlike most others in the wide domain of language processing since it involves the analysis of a very well-defined and deterministic discourse. The task of evaluating mathematical sequences, in particular, is especially open-ended. Though there has been much previous work in this domain, most approaches have utilized prior knowledge of rules or algorithms in order to evaluate input expressions. Very few methods, however, have attempted to independently *learn* these specific rules with limited amounts of supervision.

The task of unsupervised evaluation is both an intricate and subtle challenge – it often requires models to assimilate interactions between specific characters that might be separated by reasonably long intervening sequences. For more complicated operations, such as multiplication or division, successful prediction is often predicated upon assimilating interactions between pairs of digits over varying ranges, which makes it difficult for simple linearly scanning networks to approximate these operations. Attaining success on such instances will thus necessitate the development of more powerful models.

The unsupervised extraction of rules is also an integral part of human development, where we ourselves have the innate ability to abstract mathematical rules and restrictions based purely on example. Developing algorithms that can similarly extract mathematical "understanding" purely from input data is thus key towards the completion of a more refined and sensitive artificial intelligence framework.

In this project, we thus focus on the inference of mathematical rules from purely symbolic data. We pursue both discriminative and generative formulations of this challenge, and evaluate a range of computational models on a newly constructed dataset.

1.1 Previous Work

We now discuss several previous attempts to address challenges related to the task of learning-based evaluation.

Graves et al. make the assertion that Recurrent Neural Networks (RNNs) are Turing-Complete, and formulate a Neural Turing Machine that is capable of performing operations such as priority sorting [1]. They additionally state that RNNs can be used to model arbitrary operations but only *if* properly configured. Their work is thus extremely relevant to our task at hand, due to the inherent implication that these models are capable of learning and therefore representing a variety of mathematical functions.

In their recent paper *Learning to Execute*, Zaremba and Sutskever utilize Long Short-Term Memory (LSTM) networks [2] in conjunction with curriculum learning to obtain exceptional performance on the addition of two 9-digit numbers. This remarkable outcome, however, was obtained by executing LSTMs on Python-like programs as opposed to pure mathematical sequences, and inputs were constrained to computer programs that focused solely on addition. Nevertheless, the success of this approach indicates that there is much potential for experimentation with other forms of sequence inputs and operations.

In their highly influential work on deep learning [3], Hochreiter et al. demonstrated that LSTMs are remarkably suited for performing such tasks as addition, excelling under the inherent ability to selectively 'remember' certain states of the input in previous timesteps. They thus propound the structure of a new network and show that it can obtain remarkable outputs under various testing conditions. However, they note that the model does not excel in multiplication-related tasks.

Thus, all of these approaches have demonstrated the great potential for recurrent networks to act as learning-based computational models which can evaluate arbitrary input expressions. In our project, we thus seek to expand on this potential and investigate the capabilities of neural networks for learning mathematical structure from input sequences.

2 Technical Approach

In this section, we now more formally describe our challenge of mathematical evaluation. We then discuss the various models and techniques we apply to this task, and in section 3 we provide our corresponding results.

2.1 Verification

We divide mathematical evaluation into two distinct, though related, challenges. We first pursue the discriminative task of *verification*: Given an input sequence, representing a single equality or expression, we would like to determine if this expression is a mathematically valid statement. Thus, we perform binary classification on input sequences to determine if output matches the evaluation of the input. Attaining high accuracy on this task would imply knowledge of what constitutes valid operations and how those operations implicitly function.

2.2 Evaluation

The second task that we address is *evaluation*, where we are given a mathematical expression, and our goal is to obtain an evaluation of this expression. We can in fact pose this challenge as machine translation, where our aim is to translate source language s into target language t. In our case, the source language is represented by the left-hand side of a mathematical statement, and the target is simply the evaluation of this expression (or alternately an mathematically equivalent formulation). This represents a far more difficult challenge than verification – while the former requires only the implicit knowledge of mathematical understanding, evaluation necessitates that the models be able to apply these rules (or reasonably approximate them) in order to achieve success.

Discriminative	Generative
(‘176 + 300 = 878’, 0)	(‘451 + 826 - 201’, ‘1076’)
(‘321 + 71 * 124 - 1 = 1609’, 0)	(‘969 * 561’, ‘543609’)
(‘40 - 349 = -309’, 1)	(‘279 + 414 + 300’, ‘993’)
(‘372 * 372 = 138384’, 1)	(‘979 - 990’, ‘-11’)

Table 1: Examples extracted from training partitions of discriminative and generative datasets.

2.3 Data

To the best of our knowledge, there are no publicly available datasets that are well-suited to our challenge. However, due to the nature of our problem, we found that it is in fact possible to generate our own data as necessary. We thus constructed two novel datasets – one for each task. For each of our datasets, we generate expressions based on three fundamental operations: **Addition**, **Subtraction**, and **Multiplication**. Division was excluded due to the potential presence of non-terminating real numbers in the output (though we could imagine implementing a rounding strategy if we desired).

For the discriminative data, we formulate expressions that represent supposed mathematical equivalences. We ensure during the generation of this data that there are an equivalent number of positive and negative examples, where each example has a binary label describing whether it is valid or not. We generate sequences consisting purely of addition, subtraction, or multiplication, as well as input sequences that contain compositions of these operators. Every operand in a sequence is restricted to have at most 3 digits.

For the generative set, the input likewise consists of random sequences of 3-digit integers, where we once again generate data that consists purely of addition, subtraction, multiplication, or combinations of these. The target output is the respective mathematical evaluation of each sequence. See the figure below for samples from some of our datasets.

For each sequence type (addition, subtraction, multiplication, and combination) in each dataset, we generate 10000 examples for training, 2000 examples for developmental tuning, and 2000 examples for testing.

2.4 Models

We have implemented a set of models to apply to our data. We first constructed the following baseline models for experimental comparison:

- **Artificial Neural Network (ANN)** To serve as a baseline for the verification task, we construct a standard two-hidden-layer ANN with $tanh$ non-linearities and 50 nodes at each hidden dimension. Since input sequences can vary greatly in length, we sum over each of the character vectors (which we learn during training) to obtain a single-vector representation of an input expression, which we finally pass into the network. The output of the network is a simple binary label indicating whether the input equation is valid.
- **Bigram Baseline** For a baseline for the task of evaluation, we implemented a modified version of bigrams, where we take counts of the output characters that correspond to each pair of corresponding digits of inputs.

We implemented the following more nuanced models for verification and evaluation:

- **Discriminative Recurrent Neural Network (D-RNN)** The first model we implement for our task of verification is a Discriminative RNN. In this framework, we simply pass in each sequence one character at a time, and continually update our hidden state of the network. The final hidden state is then passed to a binary softmax classifier which outputs a label.
- **Discriminative Gated Recurrent Unit (D-GRU)** We build a model equivalent to that of the D-RNN, except that we now replace the network structure with a GRU. We once again utilize a binary output labeling.

- **RNN Encoder-Decoder** Our first main model for the generative task of evaluation is an encoder-decoder. As mentioned before, since our task is essentially that of machine translation, we would expect that models which perform well on translation would excel at the task of evaluation. We thus implement an RNN Encoder-Decoder with a single hidden layer at each time-step, where both encoder and decoder are standard RNNs. The final hidden state of the encoder is passed to the decoder, which outputs tokens until it reaches a stop token.

- **Long Short-Term Memory (LSTM)** Though we expect the standard Encoder-Decoder model that was described above to perform well on reasonably sized sequences, it is most likely the case that it would not generalize to longer or more complicated expressions, where vanishing or exploding gradients might become an issue. Due to the success of LSTMs at "memorizing" long inputs, we decided to implement an encoder-decoder based on this model as well. Note that we use an LSTM structure for both the encoder and the decoder (though it might certainly be reasonable to only set the encoder as an LSTM).

- **Gated Recurrent Unit (GRU)** Due to the presence of special "gates" which allow specific retention through the network [4], we would expect GRUs, as with LSTMs, to succeed especially in modelling more complicated operations or longer expressions than the standard RNN. We thus implement an encoder-decoder model with GRUs serving as both the encoder and the decoder.

- **Recurrent-Recurrent Neural Network (R-RNN)** We now make the fundamental realization that many mathematical operations can involve complex interactions between digits that are not clearly related to each other in terms of position. For example, multiplication between two numbers is a *quadratic* process, involving iterations through unique pairs of numbers, where each pair contributes to the output in a non-direct way. However, we notice that the RNN and GRU essentially represent translators that make a *linear* pass across the input data. Thus, in some sense, these linear models are perhaps innately constrained, since multiplication requires $O(n^2)$ work, where as standard recurrent networks run linearly ($O(n)$) in the input size. This suggests the fundamental need to have a model that is able to more explicitly capture these subtle interactions between inputs. We thus propose a new model to achieve exactly this power of representation, which we now call a *Recurrent-Recurrent Neural Network (R-RNN)*.

  The structure of our network is shown in the figure below. The entire intuition is that instead of directly inputting the characters of a sequence into our encoder, we can model their pairwise interactions by first putting each character through a sub-RNN that relates a given character to all characters in the sequence. That is, if s is the index of a particular character in a sequence, then we can express a single hidden state of a sub-RNN as:

  $$h_{s,t} = f(W_x \cdot L[x_t] + W_s \cdot L[x_s] + W_h \cdot h_{s,t-1})$$

  where f is a standard non-linearity (we use $tanh$ which seems to provide superior results). Thus, the term $L[x_s]$ is being incorporated at every timestep in the RNN, and is related to each other character in the sequence $L[x_t]$. Each final hidden vector then serves as the input to a *larger* RNN that serves as the main encoder for the model. Thus, for each timestep $\{1, 2, \ldots, T\}$, we have that the corresponding hidden vector h'_t of the overall network is given by:

  $$h'_t = f(W_{x'} \cdot h_t + W_{h'} \cdot h'_{t-1})$$

  Note that though we share the same set of parameters across each of the individual component-RNNs, we utilize an entirely different set of weights for our main encoder. This final hidden vector is passed into a decoder, where the decoder is a standard RNN, as described earlier.

  (Note that since our model abstracts away workings of the component recurrent networks that constitute the encoder, we could easily replace these with GRUs or LSTMs to obtain a more nuanced model, if desired.)

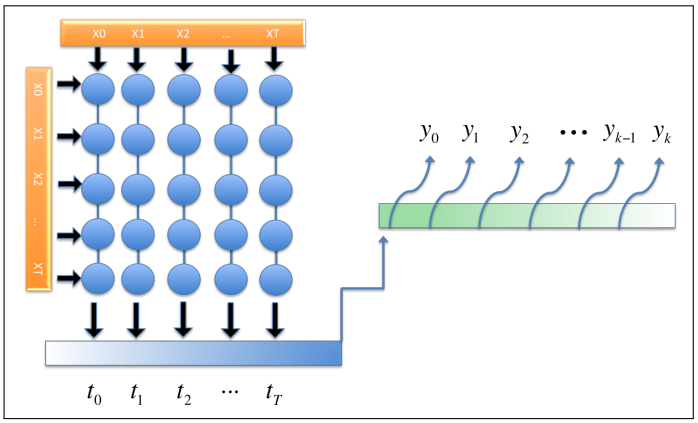


Figure 1: **The Recurrent-Recurrent Neural Network (R-RNN)**. Inputs are convolved over each other via individual sub-networks. Each of these outputs will serve as input to a larger encoder, which produces the final hidden state of an equation to pass to the decoder.

3 Experiments & Results

In this section, we now describe the experimental setup we use to evaluate our algorithms, and discuss the accompanying results.

For all of our models, we relied on no existing codebases, and instead implemented everything ourselves in NumPy and Theano to allow for more flexibility in experimentation. The entire codebase can be found on GitHub [5].

We now define the metric for the evaluation of our models: the metric A calculates what percent of individual digits were actually classified correctly:

$$A = 100\% \cdot \frac{\sum_{i=1}^{N} \text{len}(\hat{y}_i == y_i)}{\sum_{i=1}^{N} \text{len}(y_i)}$$

where $\hat{y}_i$ is the predicted sequence of digits for an example i, and y_i is the corresponding ground truth output.

The results of applying our various experiments under this evaluation metric are shown below:

Model	Train score	Dev score
Bigram baseline	**79.4%**	**74.6%**
Set of RNNs	21.0%	20.9%
Set of RNNs (selective scrambling)	52.4%	51.9%
Discriminative RNN	**90.2%**	**89.2%**

In the following tables, we give the train, followed by dev scores.

	Add		Subtr		Mult		Comp	
D-RNN	90.2%	89.2%	94.3%	93.4%	97.9%	92.6%	82.0%	80.7%
D-GRU	100.0%	95.8%	99.9%	95.6%	99.3%	81.3%	50.3%	49.7%

	Add		Subtr		Mult		Comp	
BB	76.4%	71.0%	59.1%	51.4%	42.7%	36.5%		
RNN	95.0%	86.5%	88.1%	82.8%	13.9%	13.5%	7.2%	6.5%
GRU	99.8%	97.7%	99.6%	96.6%	62.9%	51.1%	35.1%	29.6%
LSTM	99.8%	97.0%	98.2%	93.0%	41.4%	40.9%	39.7%	26.5%
R-RNN	95.3%	95.1%	98.9%	94.6%	42.0%	41.3%	40.6%	27.3%

3.1 Transfer Learning

We expect that given that there are many similarities between mathematical operations – for example, addition is conceptually a very similar operation to subtraction – that if we *share* the parameter weights between these tasks, our overall performance will improve by a considerable amount.

Thus, as a sub-experiment, we train a GRU Encoder-Decoder network separately on addition and subtraction data; we then test this network on data consisting of sequences that contain *both* addition and subtraction operators, then transfering the parameters to train on the new task for a limited number of epochs. We compare this result with another GRU model that we train without transfer learning on the data. The results are shown in the table below.

	Simpcomp		Simpcomp (transfer)	
GRU	47.2%	41.1%	38.0%	35.8%

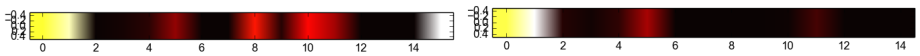
Something interesting to later investigate is the relationship between the weights learned through discrimination and those in the generative process – we would wonder if pre-training on the discriminative weights might boost performance on generative tasks – or likewise if generative models are inherently discriminative, and thus would allow them to excel in verification.

4 Analysis & Discussion

As we can see from our results, many of our models achieve exceptional performance on both the discriminative and generative tasks.

For the task of verification, our baseline model outputs reasonable performance across all data types despite being a simple model – for the process of summing over input vectors is an inherent loss of information. Our D-RNN model, on the other hand, succeeds in obtaining a significant increase in performance, reaching nearly 90% accuracy on the development set. The D-GRU framework, however, seems to excel in the areas of addition and subtraction. This is of course expected, since it is a fundamentally more powerful model than either of the other two that we implemented. Additionally, the results it obtains on the addition and subtraction data yield some very interesting insights. In the figure below, we apply T-SNE to the hidden vectors that are generated by the D-GRU before entering the softmax classifier. As we can see, there are several individual clusters that are forming, and what is remarkable is that these are grouped together by output label. Thus, the model is clearly able to learn representations of the data that are in keeping with their original structure.

We can also examine the actual "thought process" of the classifier. In the two images below, we show the result of outputting at each timestep the confidence with which the D-GRU believes that the sequence is valid. The first corresponds to the valid input "372 + 865 = 1237", while the second corresponds to the invalid example "372 + 622 = 821".



Note that, as expected, the positive input does not become purely positive until the very end of the sequence (since the equation does not actually become true until we reach the last digit). On the other hand, the invalid expression is constantly maintained as negative throughout the entirety of the forward pass.

Another interesting aspect of our verification experiments is that the standard D-RNN seems to perform more optimally on the multiplication data (and hence the composed data, which includes

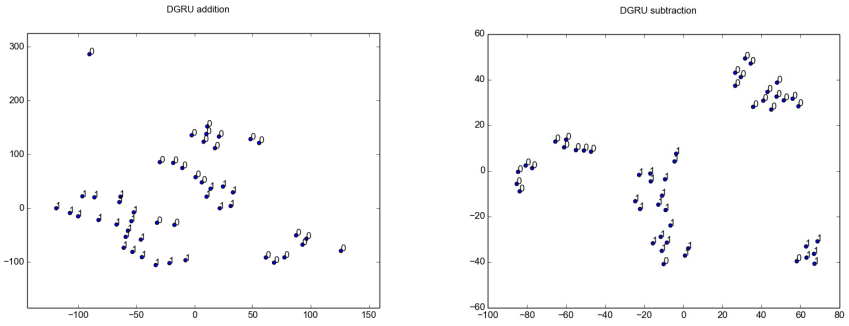


Figure 2: T-SNE representation of hidden vectors output by discriminative GRU, for addition, subtraction, and multiplication. Notice how positives (1s) and negatives (0s) cluster together.

multiplication) than the D-GRU. Though it is not entirely clear why this might be the case, it seems that perhaps the training conditions for the D-GRU might not have been entirely optimal, or the model may not be inherently suited for multiplication, though this is less likely.

For our task of evaluation, we see that all of our models generally succeed in obtaining near perfect performance for addition and subtraction, but once again, multiplication seems to be less easily modeled by the networks, as expected, since the interactions are non-linear.

Our standard RNN performs reasonably well on both addition and subtraction data. We can obtain some interesting insights int our model by examining the plots of character vectors that were learned by the model during training. In the figure below, we apply PCA to obtain the top two components of each of the character vectors.

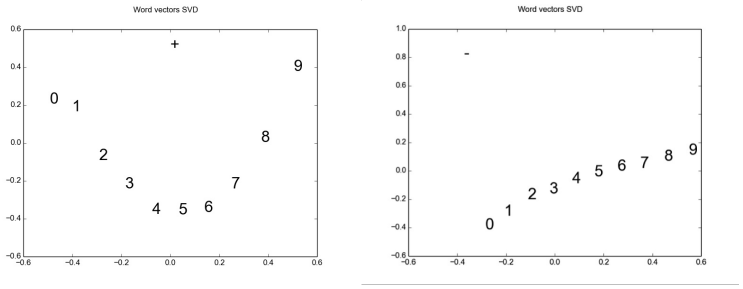


Figure 3: Word vectors learned by the RNN during training.

As we can see, the model was *independently* able to order the numbers from 0 to 9 without any prior knowledge of what the individual characters actually represented. What is also remarkable, but perhaps coincidental is that the actual shapes spanned by these characters are also inverses of each other – which is interesting since addition and subtraction are also inverses of each other. Thus, the model has clearly learned how to assimilate some of the structure of the data.

Our GRU model, however, exceeds and obtains the best performance on nearly all the given tasks, as expected. In the figures above, we obtain further interesting insight into our model by applying PCA to visualize the hidden vectors that are output by the encoder for elements in our dataset (train and dev).

For the addition data, red points are those equations with output larger than 1000 (more than 4 digits long) whereas those that are blue are equations with output less than 4 digits long. We observe a very clean separation between these two clusters, which is quite remarkable. For the subtraction data, the red points correspond to negative answers, while blue correspond to positive ones – once again, the model clearly learns the distinction between these two outputs without any prior knowledge.

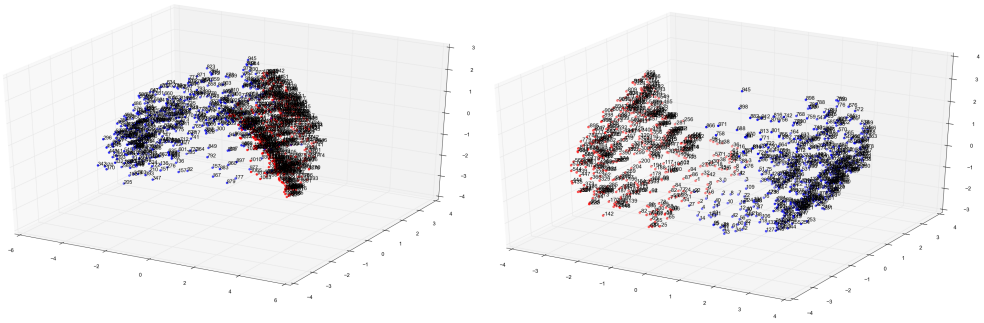
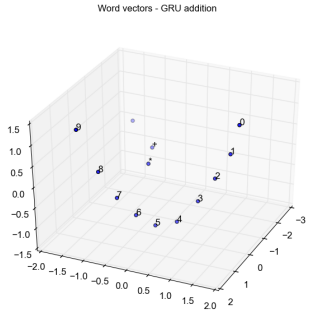


Figure 4: PCA applied to hidden vectors output by GRU for addition (left) and subtraction (right).

We also plot the word vectors for the GRU model in the figures below: notice how it too seems to learn paraboloid shapes for the vectors, as well as symmetrically order the numbers from 0 to 9 into proper shape.



GRUs, in general, seem to outperform LSTM, both in terms of computational efficiency and accuracy. It may perhaps be that LSTMs are *too* complex of a model for some of the tasks at hand, and will more likely perform better asymptotically as sequences grow longer.

For transfer learning, we would have expected an improvement as a result of applying this technique; however, because the amount of data is already relatively quite large, we might expect that the composition dataset is large enough to render such transfer unnecessary, so that learning from scratch is more beneficial.

Finally, our new model R-RNN, has shown that it is capable of achieving near equivalent performance on addition and subtraction, while still attaining boosted performance on multiplication with respect to the standard RNN. One possible reason that the performance was not as great as expected, however, is that the model is quite large and thus needs sufficient amount of training in order to converge to a more optimal solution. We suspect that the model still retains the potential for superlative performance on a variety of mathematical operations under the right conditions. Some examples that it does output correctly for multiplication are shown below.

'17 * 780 = 13260'
'705 * 77 = 54285'
'53 * 893 = 47329'

5 Conclusion

In this report, we have investigated various methods of analyzing and evaluating symbolic mathematical expressions. We have devised both discriminative and generative formulations of our task, in which we verify and evaluate arbitrary input expressions, respectively. We have also generated a novel dataset particular to our challenge that has allowed us to train a variety of recurrent models on significant amounts of data, and have additionally defined new metrics for evaluation.

Our results indicate that our simpler models are capable of achieving reasonable performance on basic operations such as addition and subtraction, though the composition of such functions as well as multiplication prove to be somewhat more challenging. Our more complex models appear to excel at the basic operations, reaching near perfect accuracies in some cases, but also face similar difficulties with multiplication, in both generative and discriminative tasks.

We thus have devised a new model, the **Recurrent-Recurrent Neural Network (R-RNN)**, that attempts to more appropriately incorporate the complex interactions associated with super-linear operations such as multiplication. Though the performance is not as superlative as hoped, the relative improvement of this framework over the other models suggests that perhaps further fine-tuning is all that is necessary to achieve desired results.

Finally, our analysis of the models' internal representations of the data indicates that the networks are, in fact, extracting some fundamental properties of numbers and mathematics – which, though expected, is quite remarkable since no information about the input sequences was assumed during training. Thus, we see our models are beginning to independently learn what rules are associated with the mathematical language.

6 Future Work

From our results, we can see that there are several directions we can pursue for the continuation of our project.

An initial goal would be to first obtain enhanced results on more complicated operations, such as multiplication and division. Though our generative models perform reasonably well on our data, there is certainly still some room for improvement; we could perhaps utilize other techniques to boost performance, such as ensembling, drop-out, or bi-directional networks.

We could then generalize to much longer or complicated expressions, by applying 1-D Convolutions or Recursive Neural Networks to our data. These models, like our R-RNN, can perhaps extract more intricate relationships between individual characters, and we would expect these models to perform well since mathematical expressions are inherently recursive.

An interesting experiment to finally pursue is the generation of mathematically equivalent expressions, as opposed to simple evaluations. Thus, the input might be "123 + 3", and the output could be (out of the many possibilities) "80 + 46". This would certainly test the generalization capabilities of the models, and would indicate a deeper understanding of the workings of mathematics.

We thus consider the challenge of learning-based evaluation to possess much potential for development, and it is our hope that the amount of subsequent work in this domain will continue to multiply over the future.

7 Acknowledgements

The authors would like to thank Jon Gauthier for offering his advice and continued support during the course of this project.

References

[1] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[2] Wojciech Zaremba and Ilya Sutskever. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.

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

[4] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. *arXiv preprint arXiv:1502.02367*, 2015.

[5] https://github.com/arthur tsang/EqnMaster.