MINIMUM DESCRIPTION LENGTH RECURRENT NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recurrent neural networks (RNNs) face two well-known challenges: (a) the difficulty of such networks to generalize appropriately as opposed to memorizing, especially from very short input sequences (generalization); and (b) the difficulty for us to understand the knowledge that the network has attained (transparency). We explore the implications to these challenges of employing a general search through neural architectures using a genetic algorithm with Minimum Description Length (MDL) as an objective function. We find that MDL leads the networks to reach adequate levels of generalization from very small corpora, improving over backpropagation-based alternatives. We demonstrate this approach by evolving networks which perform tasks of increasing complexity with absolute correctness. The resulting networks are small, easily interpretable, and unlike classical RNNs, are provably appropriate for sequences of arbitrary length even when trained on very limited corpora. One case study is addition, for which our system grows a network with just four cells, reaching 100% accuracy (and at least .999 certainty) for arbitrary large numbers.

1 INTRODUCTION

The modeling of sequential knowledge and learning requires making appropriate generalizations from input sequences that are often quite short. This holds both for language capabilities and for other sequential tasks such as counting. Moreover, it is often helpful for the modeler to inspect the acquired knowledge and reason about its properties. Neural networks, despite their impressive results and popularity in a wide range of domains, still face some challenges in these respects: they tend to overfit the learning data and require regularization or other special measures, as well as very large training corpora, to avoid this problem. In terms of knowledge, networks are often very big, and it is generally very hard to inspect a given network and determine what it is that it actually knows (see Papernot & McDaniel, 2018, among others, for a recent attempt to probe this knowledge).

Some of the challenges above arise from the reliance of common connectionist approaches on backpropagation as a training method, and in this paper we explore the implications to sequential modeling of well-known alternative perspectives on neural network design. Specifically, we consider replacing backpropagation with a general search using a genetic algorithm through a large space of possible networks using Minimum Description Length (MDL; Rissanen, 1978) as an objective function. In essence, this amounts to minimizing error as usual, while at the same time trying to minimize the size of the network. We find that MDL helps the networks reach adequate levels of generalization from very small corpora, avoiding overfitting and performing significantly better than backpropagation-based alternatives. The MDL search converges on networks that are often small, transparent, and provably correct. We illustrate this across a range of sequential tasks.

2 PREVIOUS WORK

Our work follows several lines of work in the literature. Evolutionary programming has been used to evolve neural networks in a range of studies. Early work that uses genetic algorithms for various aspects of neural network optimization includes Miller et al. (1989), Montana & Davis (1989), Whitley et al. (1990), and Zhang & Mühlenbein (1993; 1995). These works focus on feed-forward architectures, but Angeline et al. (1994) present an evolutionary algorithm that discovers recurrent

neural networks and test it on a range of sequential tasks that are very relevant to the goals of the current paper. Evolutionary programming for neural networks remains an active area of research (see Schmidhuber, 2015 and Gaier & Ha, 2019, among others, for relevant references).

In terms of objective function, Zhang & Mühlenbein (1993; 1995) use a simplicity metric that is essentially the same as the MDL metric that we use (and describe below). Schmidhuber (1997) presents an algorithm for discovering networks that optimize a simplicity metric that is closely related to MDL. Simplicity criteria have been used in a range of works on neural networks, including recent contributions (e.g., Ahmadizar et al., 2015 and Gaier & Ha, 2019).

Our paper connects also with the literature on using recurrent neural networks for grammar induction and on the interpretation of such networks in terms of symbolic knowledge (often formal-language theoretic objects). These challenges were already taken up by early work on recurrent neural networks (see Giles et al., 1990 and Elman, 1990, among others), and they remain the focus of recent work (see, e.g., Wang et al., 2018 and Weiss et al., 2018). See Jacobsson (2005) and Wang et al. (2018) for discussion and further references.

Our contribution is a complete evaluation and comparison of various classic RNN architectures against what we think is the most straightforward alternative, based on as literal as possible an application of description length principles.

3 LEARNER

3.1 MDL

Consider a hypothesis space $\mathcal{G}$ of possible grammars, and a corpus of input data D. In our case, $\mathcal{G}$ is the set of all possible network architectures expressible using our representations, and D is a set of input sequences. For a given $G \in \mathcal{G}$ we may consider the ways in which we can encode the data D given that G. The MDL principle (Rissanen, 1978), a computable approximation of Kolmogorov Complexity (Solomonoff, 1964; Kolmogorov, 1965; Chaitin, 1966), aims at the G that minimizes $|G| + |D : G|$, where $|G|$ is the size of G and $|D : G|$ is the length of the shortest encoding of D given G (with both components typically measured in bits). Minimizing $|G|$ favors small, general grammars that often fit the data poorly. Minimizing $|D : G|$ favors large, overly specific grammars that overfit the data. By minimizing the sum, MDL aims at an intermediate level of generalization: reasonably small grammars that fit the data reasonably well. MDL – and the closely related Bayesian approach to induction – have been used in a wide range of models of linguistic phenomena, in which one is often required to generalize from very limited data (see Horning, 1969, Berwick, 1982, Stolcke, 1994, Grünwald, 1996, and de Marcken, 1996, among others).

The term $|D : G|$ corresponds to the surprisal of the data D according to the probability distribution defined by G and is closely related to the cross-entropy between the distribution defined by G and the true distribution that generated D. The term $|G|$ depends on an encoding scheme for a network. We provide the details of such an encoding scheme in Appendix A and now turn to describe the space of networks that will be considered.

3.2 REPRESENTATIONS

A network is represented as a directed graph which contains nodes, weighted edges, and activation functions for each node. Since we do not use backpropagation to train the network, the set of possible networks is larger here than what is usually allowed; for example, output units can have outgoing edges which feed hidden units, and input units can feed into other input units, thus saving intermediate hidden units. Beyond the topological flexibility, the activation functions also allow for more diversity in the possible networks, they can vary freely from one unit to the next, and they can be chosen from any set of possible activation functions, including non-differentiable ones since training does not rely on backpropagation. Currently, we only allow four possible activation functions: identity (i.e., no activation function), square function, ReLU, and sigmoid.

Since we are ultimately interested in sequential tasks, we add a second type of edges – recurrent edges – which cross time steps and feed a unit with the value of another unit at the previous step. Such edges are required in order to create memory cells and counters for various sequential tasks.

3.3 SEARCH

Given our use of MDL as an objective function, which is not differentiable, and our aim of optimizing the network structure itself rather than just the weights of a fixed architecture, gradient-based training methods such as backpropagation would not naturally support this objective.

Instead, we use a Genetic Algorithm (GA; Holland, 1975) which frees us from the constraints coming from backpropagation and fulfills the two requirements at once. The GA advances by incrementally evolving networks (e.g. add an edge, adjust a weight, remove a unit, etc.), and ranks them by their MDL score. The implementation details of the GA are given in Appendix B.[1]

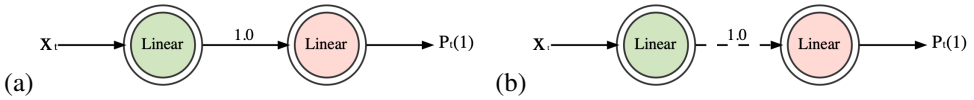
3.4 INPUT AND OUTPUT

In all tasks, the learner is fed with inputs from a sequence, one input after the next, and at each time step its outputs are interpreted as determining the probability to obtain a particular output. Depending on the task, this output is deterministically or probabilistically derivable from the inputs up to this point, and it may or may not correspond to the next input in the sequence.

If the vocabulary contains n letters, the inputs are one-hot encoded over n input cells (in green in the figures), and the outputs are given in n cells (in red). To interpret these n outputs as a probability distribution we zero negative values and normalize the rest to sum to 1. In case of a degenerate network that outputs all 0's, the probabilities are set to the uniform value $1/n$. When the vocabulary is binary, we use a single 0/1-valued input cell and a single output cell whose value is interpreted as the probability to obtain 1, clipping it to the $[0, 1]$ range if necessary.

3.5 ILLUSTRATION WITH ELEMENTARY, DETERMINISTIC TASKS

First, let us provide a simple illustration, the **identity** task: the network is fed with a random sequence of binary digits, and the target output is identical to the input at each time step. The network developed by the MDL learner is given in Fig. 1(a); this network is transparent and shows that this very simple task is learned perfectly well. Anticipating on a number of baselines, more results are given visually in Appendix D and numerically in Appendix E, where it can be seen that though some (but not all) classic RNNs may perform this task well, they do not achieve a perfect performance, only a statistically good one (that is, assigning a high probability to the appropriate value, but not necessarily assigning it a probability of 1).



(a) (b)

Figure 1: The networks found by the MDL learner for (a) the **identity** task and (b) the **previous character task** (same result for all training sets of length 10, 20, 50, and 100). The input (arriving to the green cell on the left) is directly fed into the output (red cell on the right) with a weight of 1, either through a direct, 'contemporary' connection (a), or through a cross-time connection in (b).

Second, consider the **previous character** task: the network is fed with a random sequence of binary digits, and the output is identical to the *previous* input at each time step. This task requires the learner to develop some kind of memory. The network developed by the MDL learner is given in Fig. 1(b). Again, this network is transparent, the task is learned perfectly well, and no RNNs would do as well.

4 ARTIFICIAL LANGUAGE-MODELING EXPERIMENTS

4.1 SETUP

A convenient choice to test an MDL learner and its generalization capabilities comes from language induction tasks, in which the corpus is generated from a formally well-identified generalization. We

[1]The model source code and experimental material will be published once the paper can be de-anonymized.

ran tasks based on several classical formal-language learning challenges from the linguistic domains of syntax and semantics. Let us mention two aspects in which these tasks differ from deterministic tasks as the ones above. First, they concern the prediction of the **next character in a sequence** (not about predicting an independent output based on the input). Second, the next character — that is, the target output — was **not a deterministic function** of the input (or all inputs so far), but followed in the training and test set from a more general probability distribution. Given the results above, this could be seen as a challenge for the MDL learner which seems to make categorical decisions.

4.2 BASELINES

To compete with our MDL learner, we trained 12 standard RNNs, varying in their architecture (**GRU** cells, **LSTM** cells, **Elman** cells) and the size of their hidden state vector (2, 4, 32, 128). As usual, a final softmax, derivable layer was plugged at the end of these networks for them to output a well-formed probability distribution. These RNNs were trained with a cross-entropy loss[2].

Additionally, we added two abstract baselines: a **uniform** baseline corresponding to predicting all outputs with equal probability in all cases, and an **optimal** baseline, corresponding to predicting the actual probability distribution, that is the one determined by how the task was set up.

4.3 EXPERIMENTS

In each of the tasks below the training corpora consist of several sequences of the form $s_1\#s_2\#\ldots$, where the s_i's are strings over an alphabet that does not include $\#$. The structure of the s_i's corresponds to various formal language-theoretical regularities, as described below. In sections 4.3.1 to 4.3.3 these regularities come from the domain of the semantics of quantificational determiners (Barwise & Cooper, 1981; see Tiede, 1999 and Paperno, 2011, among others, for a discussion of the learnability of such patterns). The formal languages in these tasks are regular and can be dealt with by finite-state automata. In sections 4.3.4 and 4.3.5 we consider patterns of unbounded counting based on a classic syntactic challenge (Gers & Schmidhuber, 2001) and correspond to context-free and context-sensitive languages, which require more expressive frameworks. All results are given visually in Appendix D and numerically in Appendix E.

4.3.1 EXACTLY n

In this task, each s_i has zero or more 0's, and exactly n 1's: at each time step, the next input is 0 or 1 with equal probability if the current s_i has fewer than n 1's and it is 0 or $\#$ if there are exactly n 1's already. The order of the 0's and 1's is thus random. For 'exactly 3', for example, one possible s_i would be 001011000. The model trained on sequences of length 100, 200, 500, and 1,000, and tested on an unseen sequence of length 1,000.

The MDL learner achieves a test cross-entropy of 1580, 1293, 997.2, 997.2 for the four training sets, the lowest possible cross-entropy being 997. Against the RNN alternatives, the MDL networks are ranked 2/13, 2/13, 1/13, 1/13. Fig. 2 shows the network found for $n = 1$ and the largest training set of length 1,000.

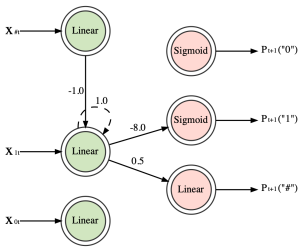


Figure 2: The network found by the MDL learner for the **exactly 1** task. The network keeps track of the number of 1's seen so far in the middle input cell (see the recurrent connection that makes it persists in memory), and resets it when a $\#$ is input (see the -1 connection from the top input cell).

[2]All RNNs were trained using the Adam optimizer (Kingma & Ba, 2015) with learning rate 0.01, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. The networks were trained by feeding the full batch of training data for 1,000 epochs.

4.3.2 AT LEAST n

In this task, each s_i has zero or more 0's, and n or more 1's: at each time step, the next input is 0 or 1 with equal probability if the current s_i has fewer than n 1's, and it is 0, 1 or # if there are exactly n 1's already, also with equal probability. The order of the 0's and 1's is thus random. For 'at least 3', for example, one possible s_i would be 01010110010.

The model trained on sequences of length 200, 500, and 1,000, and tested on an unseen sequence of length 1,000. For $n = 1$, the MDL learner achieves a test cross-entropy of 1582, 1458, 1346 for the three training sets (lowest possible cross-entropy is 1345). Against the RNN alternatives, the networks are ranked 2/13, 3/13, 1/13. Fig. 3(a) shows the network found for $n = 1$ and the largest training set.

4.3.3 BETWEEN m AND n

Here each s_i has zero or more 0's, and between m and n 1's: at each time step, the next input is 0 or 1 with equal probability if the current s_i has less than m 1's, it is 0, 1 or # if there are between m and $n - 1$ 1's already, also with equal probability, and it is 0 or 1 if the number of 1's has reached n. The order of the 0's and 1's is thus random. For 'between 3 and 6', for example, one possible s_i would be 01010110010.

This was tested with $(m, n) = (3, 6)$. The model trained on sequences of length 100, 200, 500, and 1,000, and tested on an unseen sequence of length 1,000. The MDL learner achieves a cross-entropy of 1580, 1394, 1175, 1176 for the four training sets (lowest possible cross-entropy is 1159). Against the RNN alternatives, the MDL networks are ranked 1/13, 2/13, 1/13, 1/13. Fig. 3(b) shows the network found for the largest training set.

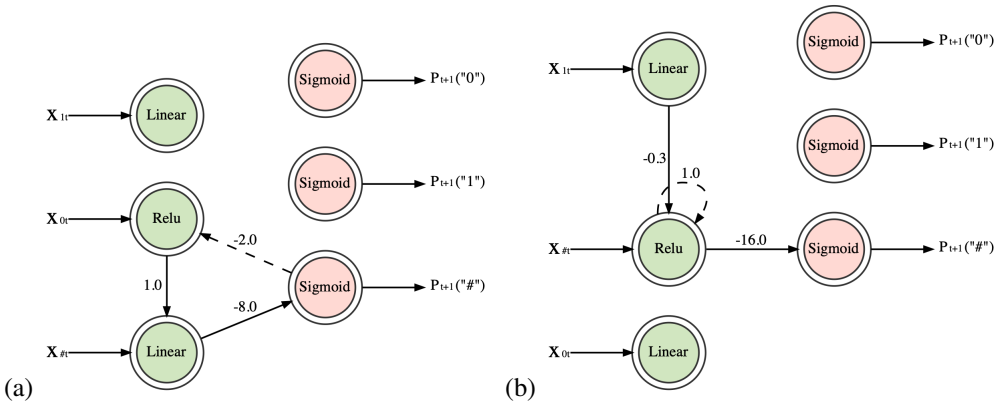


Figure 3: The network found by the MDL learner for (a) the **at least 1** task and (b) the **between 3 and 6** task for the largest training set.

4.3.4 $a^n b^n$

Here, each s_i belongs to the context-free language $a^n b^n$, where $n \geq 0$. In order to recognize the language, an unbounded counter needs to be developed.

When generating the sequences, the next character is a with probability .9; after a series of a's the sequence switches to b's with probability .1, and all remaining symbols are deterministically fixed by the fact that we aim for an $a^n b^n \#$ sequence.

The model trained on sets consisting of 10 or 100 sequences and tested on an unseen set of 1,000 sequences. The MDL learner achieves a test cross-entropy of 9829 and 4755 for the two training sets (lowest possible cross-entropy is 4680). Against the RNN alternatives, the networks are ranked 4/13 and 1/13. In Fig. 4(a) we show the network found for the larger training set.

4.3.5 $a^n b^n c^n$

Each s_i belongs to the context-sensitive language $a^n b^n c^n$, where $n \geq 0$. In order to succeed, the network needs to evolve two counters. The model trained on sets consisting of 10 or 100 sequences, randomized similarly to the previous task. Fig. 4(b) shows the network found for the larger set. This time, the model did not converge on an optimal network, but on one that balances only the a's and c's. Since previous tasks have shown that our representations make it possible to evolve counters, we attribute this failure to the search component, assuming a larger population or more generations are needed, rather than lack of expressive power. The MDL learner achieves a test cross-entropy of 14,797 and 9518 for the two training sets (lowest possible cross-entropy is 4680). Against the RNN alternatives, the networks are ranked 7/13 and 9/13.

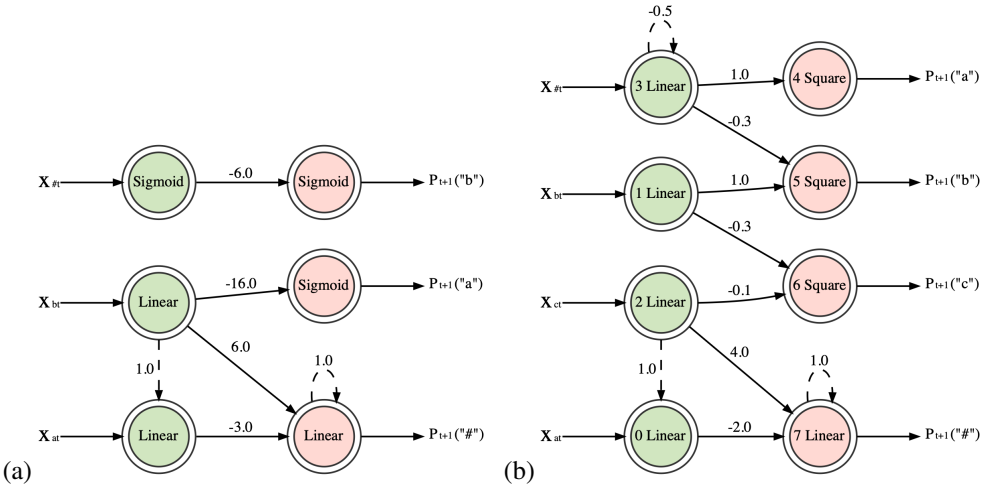


Figure 4: The network found by the MDL learner for (a) the $a^n b^n$, and (b) the $a^n b^n c^n$ task for the largest training set. Roughly, focusing on (a), the bottom right cell takes care of the counting: it decreases at each new input a (by increments of 3), and decreases at each new increment of b (by increment of 3: +6 directly from the middle input cell and -3 indirectly through the bottom left input cell).

4.4 Overview of the results

Task	Optimal cross-entropy	MDL cross-entropy	Best RNN, no. hidden units	Best RNN cross-entropy	MDL rank vs. RNN
Exactly 1	997	997.2	Elman, 2	1009	1/13
At least 1	1345	1346	Elman, 2	1366	1/13
Between 3-6	1159	1176	GRU, 2	1178	1/13
$a^n b^n$	4680	4755	LSTM, 4	4805	1/13
$a^n b^n c^n$	4680	9518	LSTM, 128	4830	9/13
Addition	0	173	Elman, 4	9050	1/13

Table 1: Results overview: cross-entropy and ranking of the MDL learner compared to the RNN alternatives (here for the largest training set in each task, see details in Appendices D and E).

Table 1 provides an overview of the results, full details are given visually in Appendix D and numerically in Appendix E. As illustrated in the figures above, the networks that the MDL learner finds are sufficiently **small and transparent** that their workings can be inspected directly. In each case, this network expresses a pattern that is either identical to the one that was used to generate the corpus or is very close to it.

Not surprisingly, this translates to **good performance** in terms of cross-entropy. Even though the learner did not attempt to optimize cross-entropy directly, the cross-entropy of the MDL network is close to the entropy of the true distribution across several corpus sizes. Sometimes the two are

almost the same, but even when this is not the case the MDL network performs no worse (and typically much better) than the random baseline.

Things are different with the comparison RNNs from the literature. These networks are large and opaque, and they perform unreliably: occasionally one of them performs well for a particular corpus size, but others will typically perform much worse than chance, and which architecture does what can change significantly for the next corpus size or the next task. Overall, the MDL learner performs best on the test set in 21 of the 40 tasks and training conditions presented here,[3] while the next best learner wins only 7 of the remaining tasks.

To the extent that we can identify a trend in the performance of the RNNs it is that the best performance generally comes from small networks, with few hidden units (for 26 tasks out of 40, the winner among the RNNs has 2 hidden units, the minimal number possible here). Smaller networks may perform worse than bigger ones on the training corpus, but they generalize better and perform better at test.[4] This is of course in line with the intuition behind our own learner and the MDL approach more broadly.

5 CASE STUDY: GENERAL ADDITION

Recent advancements in large-scale language models such as GPT-3 (Brown et al., 2020) have brought renewed attention to the capability of such models to generalize as opposed to memorize. One particular test case is that of general addition, a task which humans tackle with relative ease and using few examples, but that is not picked up in full generality by any deep learning learner to our knowledge.[5,6]

In the setting we explore, the network is fed with sequences of pairs of binary digits, representing the digits of two binary numbers to be added up. The output at each step is the corresponding digit of the sum of these two numbers. With a small training set of all pairs of integers up to 10 (total 100 samples), the MDL learner fails because in some cases it predicts a categorical probability (a plain 0 or 1) for the wrong output. With a larger training set of 400 samples (all pairs up to 20), the MDL learner develops the network given in Fig. 5. The network achieves 100% accuracy on any test set, with cross-entropy 173 compared to an optimal 0.

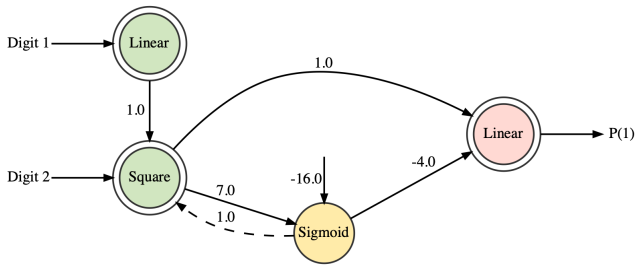


Figure 5: The network found by the MDL learner for the **addition** task, trained on 400 pairs of numbers. The first digit is added to the second digit, and the sum is squared in place. Next, a hidden cell (in yellow) with a recurrent connection was evolved to take care of the carry-over. The network reaches 100% accuracy on a test set consisting of all pairs of numbers up to 250, and is in fact provably correct for any arbitrary pair of numbers.

[3]Even in cases where the MDL learner is not the winner its cross-entropy is close to that of the winner or to the optimal baseline; in fact, sometimes the RNN winner has a cross-entropy below that of the optimal baseline, which makes it a suspicious winner.

[4]As a result, it means that picking the right RNN architecture is not an easy task (without a tripartite training/dev/test set), and performance at training is not a good predictor of performance at test.

[5]In other people's words: "As far as I know there is no neural network that is capable of doing basic arithmetic like addition and multiplication on a large number of digits based on training data rather than hard-coding.", Kevin Lacker, 'Giving GPT-3 a Turing Test'.
https://lacker.io/ai/2020/07/06/giving-gpt-3-a-turing-test.html

[6]GPT-3, one of the most recent and ambitious models, succeeds at many tasks but not quite addition: "One particularly interesting case is arithmetic calculation: the model gives a perfect score for 2-digit addition and subtraction (100% accuracy), but fails to do five digits (less than 10% accuray)." Chuan Li, 'Demystifying GPT-3'. https://lambdalabs.com/blog/demystifying-gpt-3/

Here again, this network is quite transparent. In short, the output h_n of the hidden cell (in yellow) at any given time step n corresponds to the carry-over. (With i_n and j_n the inputs, $h_n = \text{sigmoid}(7(i_n + j_n + h_{n-1})^2 - 16)$, and this goes to 1 — that is, there is a carry-over — if the sum of the inputs and the carry-over from the previous time step is large enough). In Appendix C, we show more precisely that the network assigns a probability of .999 or above to the correct output under all circumstances.

Again, the task is learned very well and in a readable fashion. None of the comparison RNNs that we consider do as well, coherent with observations made above. To our knowledge no other RNN has been proven to hold a carry-over in memory for an unbounded number of digits, i.e. to perform general addition of any arbitrary pair of numbers.

6 Conclusion

We presented a learner, building on several different lines of work in the literature, that traverses a complex space of RNNs varying in both weights and architectures, in search of the network that has the minimal description length. We tested our learner on a range of sequential tasks and compared it to various RNNs from the literature. We found that our learner arrived at networks that are reliably close to the true distribution across tasks and corpus sizes. In fact, in several cases the networks achieved perfect scores. Moreover, the networks lent themselves to direct inspection and showed an explicit statement of the pattern that generated the corpus. The RNNs from the literature, on the other hand, were not just opaque but also generally performed much less reliably on the test corpora.

In current work we attempt to extend the present paper in several directions. For example, we are extending the range of generating patterns for our corpora, including dependencies from linguistic domains not considered here, such as phonotactic patterns. We are also considering training corpora that are more challenging than the ones used here in terms of size (using training corpora that are even smaller than the ones used here), noise (corrupting the training corpus using various noise patterns), and generating distribution (deviating from the simple generating distributions used here, which supported a direct comparison of simulation results with the true distribution but are overly simplistic). We are also working on extending the learner in terms of allowable units and connections. An obvious question is whether the GA search can be sufficiently efficient to support MDL, at least on relatively small corpora; we take the current results as encouraging in in this regard. More formally, we estimate that the computational power needed to train an MDL learner should not exceed that of a regular RNN through backpropagation. [7]

Beyond these technical extensions, we are interested in connecting the present work more tightly with experimental results on parallel human biases and generalization preferences. MDL predicts very inclusive generalizations for small training corpora, with a narrowing as the corpus grows. Non-regularized RNNs do not make this prediction, and standard regularization schemes, which lower the values of weights but not the number of units, still predict time courses for generalization that differ from those of MDL. Another way to put it is: different learners come with different biases, biases which ought to be more visible with small training sets, and one question is how these biases relate to those of human learners. This question can be asked experimentally by looking at how human subjects generalize from very small corpora (see, e.g., Xu & Tenenbaum, 2007 for such a comparison in a slightly different setting). Beyond the comparison of MDL with alternatives that do not rely on a similar balance between simplicity and goodness of fit, we would like to explore a more detailed kind of comparison within the family of MDL models. Since the MDL score depends on the primitives that are provided and their given costs, it is possible to reason about different choices of primitives and costs in view of human generalization (see Piantadosi et al., 2016).

References

Panagiotis Adamidis. Review of parallel genetic algorithms bibliography. *Aristotle Univ. Thessaloniki, Thessaloniki, Greece, Tech. Rep*, 1994.

[7] Our MDL learner is trained over 1000 generations, each with 2000 individuals, with a single feedforward pass for each. Overall, this represents 2M feedforward passes, with no backpropagation step and presumably significantly smaller networks (so, faster feedforward passes).

Fardin Ahmadizar, Khabat Soltanian, Fardin AkhlaghianTab, and Ioannis Tsoulos. Artificial neural network development by means of a novel combination of grammatical evolution and genetic algorithm. *Engineering Applications of Artificial Intelligence*, 39:1–13, 2015.

P.J. Angeline, G.M. Saunders, and J.B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(1):54–65, 1994.

Jon Barwise and Robin Cooper. Generalized quantifiers and natural language. *Linguistics and Philosophy*, 4:159–219, 1981.

Robert C. Berwick. *Locality Principles and the Acquisition of Syntactic Knowledge*. PhD thesis, MIT, Cambridge, MA, 1982.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. *arXiv:2005.14165 [cs]*, July 2020. URL `http://arxiv.org/abs/2005.14165`. arXiv: 2005.14165.

Erick Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis*, 10(2):141–171, 1998.

Gregory J. Chaitin. On the length of programs for computing finite binary sequences. *Journal of the ACM*, 13:547–569, 1966.

Carl de Marcken. *Unsupervised Language Acquisition*. PhD thesis, MIT, Cambridge, MA, 1996.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

Adam Gaier and David Ha. Weight agnostic neural networks. *CoRR*, abs/1906.04358, 2019.

F.A. Gers and E. Schmidhuber. LSTM recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340, November 2001. ISSN 1941-0093. doi: 10.1109/72.963769. Conference Name: IEEE Transactions on Neural Networks.

C. Lee Giles, Guo-Zheng Sun, Hsing-Hen Chen, Yee-Chun Lee, and Dong Chen. Higher order recurrent networks and grammatical inference. In D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems 2*, pp. 380–387. Morgan-Kaufmann, 1990.

David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms*, volume 1, pp. 69–93. Elsevier, 1991.

V Scott Gordon and Darrell Whitley. Serial and parallel genetic algorithms as function optimizers. In *ICGA*, pp. 177–183, 1993.

Peter Grünwald. A minimum description length approach to grammar inference. In Stefan Wermter, Ellen Riloff, and Gabriele Scheler (eds.), *Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language Processing*, Springer Lecture Notes in Artificial Intelligence, pp. 203–216. Springer, 1996.

John H Holland. Adaptation in natural and artificial systems. an introductory analysis with application to biology, control, and artificial intelligence. *Ann Arbor, MI: University of Michigan Press*, pp. 439–444, 1975.

James Horning. *A Study of Grammatical Inference*. PhD thesis, Stanford, 1969.

Henrik Jacobsson. Rule extraction from recurrent neural networks: A taxonomy and review. *Neural Computation*, 17(6):1223–1263, 2014/07/01 2005.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference of Learning Representations (ICLR)*, 2015. URL `https://arxiv.org/abs/1412.6980`.

Andrei Nikolaevic Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission (Problemy Peredachi Informatsii)*, 1:1–7, 1965.

Ming Li and Paul Vitányi. Chapter 1.4, Binary Strings. In *An Introduction to Kolmogorov Complexity and Its Applications*, Texts in Computer Science. Springer New York, New York, NY, 2008. ISBN 978-0-387-33998-6 978-0-387-49820-1. doi: 10.1007/978-0-387-49820-1. URL http://link.springer.com/10.1007/978-0-387-49820-1.

Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. *Designing Neural Networks using Genetic Algorithms.*, volume 89. 1989.

David J Montana and Lawrence Davis. Training feedforward neural networks using genetic algorithms. In *IJCAI*, volume 89, pp. 762–767, 1989.

Denis Paperno. Learnable classes of natural language quantifiers: Two perspectives. Ms., UCLA, December 2011.

Nicolas Papernot and Patrick McDaniel. Deep k-Nearest Neighbors: Towards Confident, Interpretable and Robust Deep Learning. *arXiv:1803.04765 [cs, stat]*, March 2018. URL http://arxiv.org/abs/1803.04765. arXiv: 1803.04765.

Steven T. Piantadosi, Joshua B. Tenenbaum, and Noah D. Goodman. The logical primitives of thought: Empirical foundations for compositional cognitive models. *Psychological review*, 123 (4):392–424, 07 2016.

Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

Jürgen Schmidhuber. Discovering neural nets with low Kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873, 1997.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61(0): 85–117, 2015.

Ray J. Solomonoff. A formal theory of inductive inference, parts I and II. *Information and Control*, 7(1 & 2):1–22, 224–254, 1964.

Andreas Stolcke. *Bayesian Learning of Probabilistic Language Models*. PhD thesis, University of California at Berkeley, Berkeley, California, 1994.

Hans-Joerg Tiede. Identifiability in the limit of context-free generalized quantifiers. *Journal of Language and Computation*, 1(1):93–102, 1999.

Qinglong Wang, Kaixuan Zhang, Alexander G. Ororbia II, Xinyu Xing, Xue Liu, and C. Lee Giles. An empirical evaluation of rule extraction from recurrent neural networks. *Neural Computation*, 30(9):2568–2591, 2020/06/12 2018.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

D Whitley, T Starkweather, and C Bogart. Genetic algorithms and neural networks: optimizing connections and connectivity. *Parallel Computing*, 14(3):347–361, 1990.

Fei Xu and Joshua B. Tenenbaum. Word learning as Bayesian inference. *Psychological review*, 114 (2):245–272, 2007.

Byoung-Tak Zhang and Heinz Mühlenbein. Evolving optimal neural networks using genetic algorithms with Occam's Razor. *Complex Systems*, 7(3):199–220, 1993.

Byoung-Tak Zhang and Heinz Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38, 2020/07/11 1995.

A NETWORK ENCODING

A network consists of U units, U activation functions specified for each unit, and C connections (including bias connections). In order to represent a network as a binary string, the following serialization scheme is used.

A.1 NODES

The number of nodes in the network affects its overall encoding length both explicitly and implicitly when it is used to encode other components: node numbers are used when specifying connection sources and targets, and so larger numbers require more space; and more nodes require more activation functions to be specified.

Since the number of nodes varies from network to network, their string representation cannot be of fixed length. To ensure unique readability of a network from its string representation we use a prefix-free code.

Here and throughout this section we encode integers into bit-strings using the prefix-free encoding from Li & Vitányi (2008):

$$E(n) = \underbrace{\underbrace{11111...1111}_{\text{Unary encoding of } log_2 n} \quad \underbrace{0}_{Separator} \quad \underbrace{1010...010}_{log_2 n}}_{n}$$

Thus for an integer n its encoding length would be $2\lceil log_2 n \rceil + 1$, and the total encoding length for all units in a network would be $U(2\lceil log_2 n \rceil + 1)$.

A.2 ACTIVATIONS

For a set of A possible activation functions and U units, the encoding cost for specifying all activations is $U\lceil log_2 A \rceil$. Since A is constant throughout the simulation no prefix-free encoding is needed.

A.3 WEIGHTS

To simplify the representation of weights in neural networks and to make it easier to mutate weights incrementally in the genetic algorithm, we represent each weight as a fraction made of a sign (plus/minus) and integer numerator and denominator:

$$\pm \frac{N}{D}$$

This can be serialized into bits using the following conversion and the integer encoding scheme presented above. For example, the weight $w_{ij} = +\frac{2}{5}$ would be represented as:

$$\underbrace{\underbrace{1}_{+} \underbrace{E(2) = 10...10}_{2} \underbrace{E(5) = 1110...11}_{5}}_{w_{ij}}$$

And its encoding length $|w_{ij}|$ would be the sum of:

- 1 bit for the sign.
- $2\lceil log_2 N \rceil + 1$ bits for the numerator
- $2\lceil log_2 D \rceil + 1$ bits for the denominator.

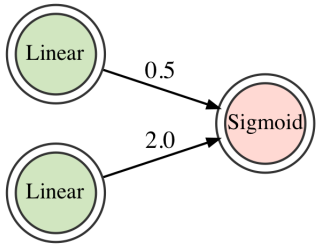
A.4 CONNECTIONS

A connection c_{ij} consists of a source unit i, a target unit j and a weight w_{ij}. It can thus be encoded as:

$$\underbrace{E(i)E(j)\underbrace{0/1E(N_{ij})E(D_{ij})}_{w_{ij}}}_{c_{ij}}$$

A.5 EXAMPLE

We'll encode the following network which consists of three units, two connections, and two weights: $\frac{1}{2} = 0.5$ and $\frac{2}{1} = 2.0$.



The final representation for this network is:

$$\underbrace{\underbrace{11011}_{E(3)}\underbrace{00}_{lin}\underbrace{00}_{lin}\underbrace{10}_{sig}\underbrace{00..11}_{c_{0,2}}\underbrace{01..01}_{w_{0,2}}\underbrace{101..011}_{c_{1,2}}\underbrace{0..11}_{w_{1,2}}}_{\text{Encoded Network}}$$

Note that $E(U)$ is prefixed to the string to make it possible to parse the activation part correctly.

B GENETIC ALGORITHM

The genetic algorithm implementation for our model comprises three main components: a population representation scheme, a selection scheme and a recombination scheme. We describe here the implementation choices made for each component.

The algorithm is initialized by creating a population of N random neural networks. Each network is initialized by randomizing the following parameters: number of hidden units, activation function for each unit, the set of forward and recurrent connections between the units, and the weights of each connection. In order to avoid an initial population that contains mostly degenerate (specifically, disconnected) networks, output units are forced to have at least one incoming connection from an earlier unit.

The number of hidden units and the weight numerators and denominators are randomized from a geometric prior ($p = 0.5$) to reflect their fitness based on the MDL metric.

The algorithm is run for g generations, where each generation involves a selection step followed by a recombination step. During selection, networks compete for survival on to the next generation based on their fitness. A network's fitness is the inverse of its MDL score. We use Tournament Selection (Goldberg & Deb, 1991), a selection method which approximates exhaustive ranking of the population by selecting random subsets of the population of size t and then selecting the best

individual from each subset; this process is repeated N times to select a new population with repetitions. The approximate nature of tournament selection prevents premature convergence by allowing less-than-optimal individuals to survive, and at the same time alleviates the computational load of the selection step by lowering its run time from $O(NlogN)$ to $O(N)$.

Out of the selected population, N offspring networks are created through either mutation or crossover with other networks. A network is mutated using one of the following operations:

1. Add/remove a unit.
2. Add/remove a forward or recurrent connection between two units.
3. Mutate a connection by incrementing/decrementing its weight's numerator or denominator, or by flipping its sign.
4. Replace a unit's activation function with another.

These mutations make it possible to grow networks and prune them when necessary, and to potentially reach any architecture that can be expressed using our building blocks.

Two networks can also be crossed-over to create an offspring network, potentially allowing networks who perform well on different aspects of the task to share their 'genes' and create a network that performs as well as its two parents combined. We cross-over two parent networks by constructing a network which feeds the two networks in parallel and averages their outputs. While this creates a larger network which is penalized by the $|G|$ term, this has the potential to create an offspring which performs better than its parents on the $|D : G|$ term and can be pruned later. Networks are randomly selected for mutation and crossover by probabilities $p_{mutation}$ and $p_{crossover}$.

On top of the basic genetic algorithm we use the Island Model (Gordon & Whitley, 1993; Adamidis, 1994; Cantú-Paz, 1998) which divides a larger population into 'islands' of equal size N, each running its own genetic algorithm as described above. Once every $m_{interval}$ generations, a *migration* step occurs during which the m_{ratio} top networks of each island are sent to another island in a round-robin fashion. At the receiving island, the lowest-ranking networks are replaced with the incoming migrants. The island model makes it possible to parallelize the algorithm by running each island on a different processor, while also mitigating against premature convergence, which often occurs when using large populations.

The simulation ends when all islands complete g generations and the best network from all islands is taken as the solution.

All simulations reported in this paper use the following hyper-parameters[8]:

- $N = 1,000$
- $islands = 72$
- $p_{mutation} = 0.9$
- $p_{crossover} = 0.1$
- $g = 1,000$
- $t = 4$
- $m_{ratio} = 0.1$
- $m_{interval} = 20$

C PROOF THAT THE NETWORK FOUND FOR ADDITION IS ACCURATE

Consider the network in Fig. 3. Call i_n and j_n the inputs at a time step n, h_n the output of the hidden cell (in yellow) and o_n the output of the output cell.

At every time step n, (i) h_n is the carry-over (0 or 1), with a margin for error for this carry-over of $\epsilon_{\text{co}} = .0013$ (that is, h_n is in $[0, \epsilon_{\text{co}}]$ if the carry-over is 0, and in $[1 - \epsilon_{\text{co}}, 1]$ if the carry-over is 1; (ii) o_n is correct with a margin of error $\epsilon = .001$, that is o_n is below .001 if the nth digit of the sum is 0, and above .999 if it is 1.

[8] All simulations ran on AWS c5.18xlarge machines with 72 vCPUs each (3.0 GHz Intel Xeon).

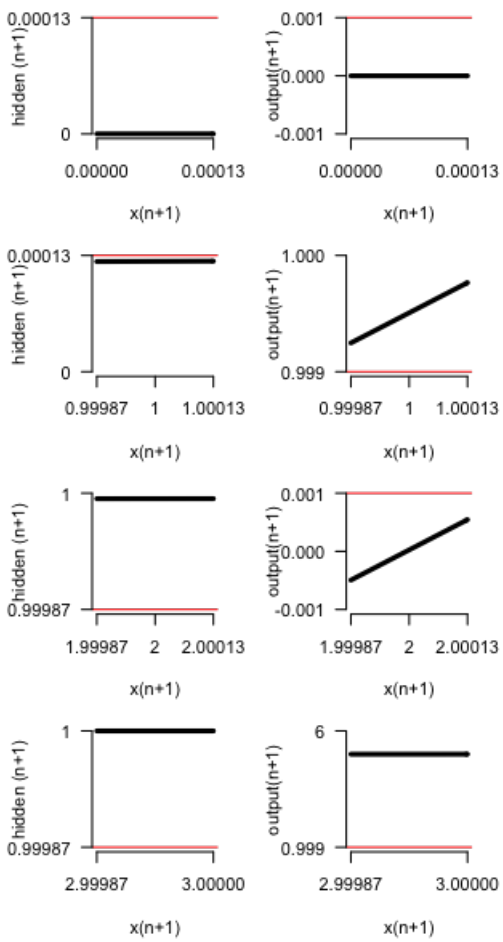
Proof. Note first that the network is such that:

- $h_n = \text{sigmoid}(7{x_n}^2 - 16)$,
- $o_n = {x_n}^2 - 4\text{sigmoid}(7{x_n}^2 - 16)$,
- with $x_n = i_n + j_n + h_{n-1}$.

From there, the theorem is proven by induction. The initialization step can be easily checked (NB: h_{-1} is set to 0 by convention). Suppose the result holds for n. Then: $x_{n+1} \in [0, \epsilon_{\text{co}}] \cup [1 - \epsilon_{\text{co}}, 1 + \epsilon_{\text{co}}] \cup [2 - \epsilon_{\text{co}}, 2 + \epsilon_{\text{co}}] \cup [3 - \epsilon_{\text{co}}, 3]$.

The fact that the result holds at the next time step $n + 1$ can be proven graphically (it can also be proven analytically, e.g., using the continuity and local monotonicity of the relevant functions):

- The left hand side of the following graph shows that h_{n+1} stays within error margin $\epsilon_{\text{co}} = .00013$, and takes the appropriate value: in binary notation, the carry-over should be 0 if x_{n+1} (the sum of the current inputs and the carry-over) is 0 or 1, and should be 1 if x_{n+1} is 2 or 3.
- The right hand side shows that o_{n+1} is correct and stays within the error margin $\epsilon = .001$: o_{n+1} should be 0 if x_{n+1} is 0 or 2 (a null unit digit here because 2 is 10 in binary notation), and 1 if x_{n+1} is 1 or 3 (in binary notation: 1 and 11).



□

D CROSS-ENTROPY

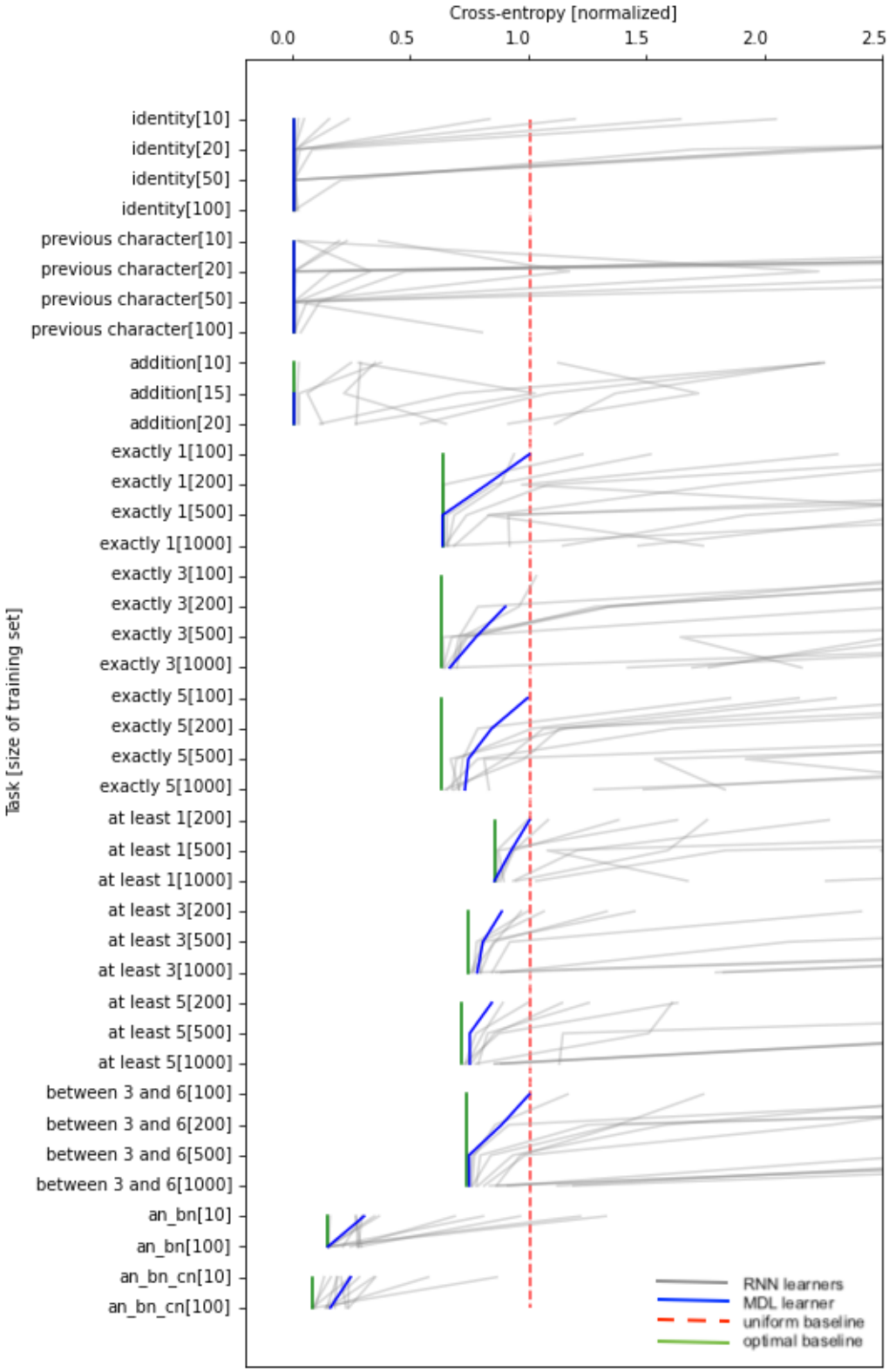


Figure 6: Cross-entropy of different learners across all the tasks presented here. For easier visualization, the cross-entropy is normalized (divided) by the cross-entropy of a learner predicting uniform probability for all outputs. At the other extreme, the green line represents an optimal baseline as the cross-entropy of a learner which would have captured the underlying task perfectly well. The gray lines represent the various RNN competitors, and the blue line is the MDL learner. The x-axis distributes the various tasks (identity, previous character, addition, exactly n, at least n, between m and n, $a^n b^n$, $a^n b^n c^n$), for increasing sizes of the training set. Corresponding numbers are given in Appendix E.

E FULL RESULTS

Corpus	Network Units / Seq. length	Elman cells 2	Elman cells 4	Elman cells 32	Elman cells 128	GRU cells 2	GRU cells 4	GRU cells 32	GRU cells 128	LSTM cells 2	LSTM cells 4	LSTM cells 32	LSTM cells 128	MDL learner	optimal baseline	uniform baseline
identity	10	18.92	42.86	5422	9557	1189	1637	2041	828	149	232	3964	8509	**0.00**	0	1000
	20	4.89	1.50	2221	2152	3.16	1.87	77.60	25.69	9.46	4.00	2366	1692	**0.00**	0	1000
	50	4.30	1.40	0.50	0.20	1.13	1.77	0.40	0.17	8.48	3.28	0.18	199.78	**0.00**	0	1000
	100	4.29	1.40	0.72	0.03	17.94	1.81	0.10	0.04	8.76	3.26	0.11	0.54	**0.00**	0	1000
previous character	10	20.80	223.79	7934	11718	13.60	190.95	7522	8150	4850	360.39	9696	9393	**0.00**	0	1000
	20	324.2	1.71	0.17	3418	2222	0.37	0.81	5.98	473.5	1166	157.7	5148	**0.00**	0	1000
	50	12.68	1.83	0.05	54.38	7.78	0.60	0.03	0.02	109.0	1.40	0.10	0.14	**0.00**	0	1000
	100	8.51	1.89	0.05	0.03	796.6	1.85	0.03	0.00	28.94	1.28	0.06	0.01	**0.00**	0	1000
addition	10	170772	**12995**	1402159	1390416	150962	176520	1403422	5353445	214554	230052	700352	5094945	∞	0	625000
	15	637908	8794	673833	851466	35499	169586	435626	2112948	132403	14379	1070754	2894081	**211**	0	625000
	20	164313	9050	335796	691196	75528	164883	68920	2286764	401935	14029	567814	2554335	**173**	0	625000
exactly 1	100	1937	3648	7120	7034	2394	5071	6541	7407	**1477**	5064	7565	8919	1580	997	1584
	200	**1004**	1530	8508	6538	1338	1701	6225	6931	1386	2885	5237	6022	1293	997	1584
	500	1004	5193	1303	1439	1028	1159	4021	5768	1076	1310	4226	3034	**997.2**	997	1584
	1000	1009	1034	2745	1447	1027	1012	4397	4653	1033	1076	2313	1806	**997.2**	997	1584
exactly 3	100	**1626**	4340	5833	4689.50	6806	5248	7476	6501	4790	5416	6002	6310	∞	995	1585
	200	1510	2108	8623	7510	4070	2124	6391	8421.04	**1234**	2024	4802	7856	1419	995	1585
	500	1125	1059	2596	7156	**1000**	1152	4016	6294	1109	1165	4354	6649	1221	995	1585
	1000	1095	**999**	3407	1070	1007	1016	2786	6627	1063	1048	2676	2240	1046	995	1585
exactly 5	100	2928	3388	9085	7559	3636	6239	8463	8184	4526	6224	7863	15394	**1568**	989	1585
	200	**1234**	1598	9447	7054	1783	1676	6858	8684	1825	2543	8953	8205	1327	989	1585
	500	1133	1090	5924	1277	1544	1417	3032	6049	**1051**	1233	2424	7073	1170	989	1585
	1000	1107	1062	2348	1310	**1013**	1077	5110	5405	1109	1027	2892	2017	1148	989	1585
at least 1	200	**1569**	2572	11890	8643	1705	3591	8630	9529	2178	2773	8013	8405	1582	1345	1585
	500	**1364**	1373	2895	1701	1476	1899	6358	8643	1469	2509	3939	8420	1458	1345	1585
	1000	1366	1409	1626	2642	1385	1473	6044	7393	1367	1491	5930	3572	**1346**	1345	1585
at least 3	200	1568	1525	5818	8694	2285	2099	7431	8502	1677	3807	5876	7219	**1395**	1168	1585
	500	**1226**	1351	7794	3320	1275	1340	6996	8227	1253	1449	7400	7913	1269	1168	1585
	1000	1206	1195	1360	1393	**1186**	1254	6000	2884	1255	1330	1232	2833	1230	1168	1585
at least 5	200	1402	1800	8425	8175	1569	1979	10265	8374	2573	2539	4720	7576	**1327**	1128	1585
	500	1270	1346	1806	5142	1223	1300	5251	7394	1376	2381	5308	6470	**1180**	1128	1585
	1000	1164	1186	1781	1350	1149	1238	1346	7467	**1145**	1219	1392	5466	1180	1128	1585
between 3 and 6	100	1837	2747	8996	7174	5573	5079	7825	10198	9289	4839	7211	9592	**1580**	1159	1585
	200	**1351**	2172	2191	8664	1443	3614	7122	7466	1973	2497	6668	9132	1394	1159	1585
	500	1212	1322	1521	6612	1198	1552	7369	4212	1251	1469	5586	6234	**1175**	1159	1585
	1000	1195	1198	1359	1316	1178	1276	1876	1435	1224	1236	1769	1370	**1176**	1159	1585
an_bn	10	8634	8727	9589	22531	9527	31676	40108	43700	**5057**	11908	26510	11257	9829	4680	33085
	100	9431	8902	6872	7941	9012	9092	4832	4960	4865	4805	5047	4841	**4755**	4680	33085
an_bn_cn	10	21242	12297	17047	20829	14321	11392	35052	52944	11159	**7867**	9625	14884	14797	4680	61622
	100	14303	13558	8158	9078	14119	8533	8786	5563	10693	5485	4873	**4830**	9518	4680	61622

Table 2: Cross-entropy of different learners across all the tasks presented here. The winner with the lowest cross-entropy (without considering the high and low baselines in the last two columns, of course) is in bold for each task (row).