We explore a neural network architecture that stacks a recurrent layer and a feedforward layer that is also connected to the input, and compare it to standard Elman and LSTM architectures in terms of accuracy and interpretability. When noise is introduced into the activation function of the recurrent units, these neurons are forced into a binary activation regime that makes the networks behave much as finite automata. The resulting models are simpler, easier to interpret and get higher accuracy on different sample problems, including the recognition of regular languages, the computation of additions in different bases and the generation of arithmetic expressions.

1 INTRODUCTION

Machine learning techniques, and more specifically deep neural networks (DNNs), have become essential for a wide range of applications, such as image classification [1, 19], speech recognition [12] or natural language processing [4, 20, 26]. Notwithstanding, nowadays these deep models are still not dominant in many applications due to the common belief that simpler approaches, such as linear models or decision trees, provide better interpretability. Hence techniques for interpreting DNNs are becoming popular in fields like image classification [21] and sequence modeling including music composition and natural language generation [18, 25]. However, and in spite of all the effort, the interpretation and understanding of DNNs is still an open question that deserves further research.

Recurrent Neural Networks (RNNs) are a kind of deep network aimed at sequence modeling, where the depth comes from a recurrent loop in the network architecture that forces a backpropagation through several time steps when the gradients are computed [11]. Since their introduction, RNNs have been shown to be Turing equivalent [24], and many authors have studied the ability of these networks to model different kinds of formal languages [9, 23, 29]. The interpretability of RNNs has been often addressed by quantization approaches that try to reduce the network to a set of rules (rule extraction) [17, 23, 28], usually in the form of a deterministic automaton [3, 6, 10].

More recently, the picture has been completed with the introduction of Memory Augmented Neural Networks (MANNs) [7], where a standard, and usually recurrent, neural network is enhanced with some type of external memory [2, 13, 14]. Results of these new models on complex problems seem very promising. Additionally, as much of the computational power of these networks relies on the memory, the complexity of the neural component is reduced, hence also improving the overall model interpretability. The connection with general automata seems obvious in this case, with the neural network implementing a kind of finite state neural processor and different memory schemes leading to different types of abstract models, from pushdown automata to complete Turing machines.

Following these ideas, in this article we explore a neural network architecture that combines a simple feedforward processing layer with a recurrent layer that implements a sort of memory. When this Dual network is trained to process temporal sequences, the recurrent layer is used to keep track of only the essential information that must be preserved along time, without performing any additional computation. The feedforward layer combines in turn the input and the memory content to provide the final network’s output. This separation of roles seems to be beneficial for learning, since much of the computational power is discharged from the recurrent connection, and at the same time improves interpretability. From a more abstract point of view, we find that the network may be reduced to a Mealy machine, just as a simple recurrent network can be reduced to a finite automaton in the form of a Moore machine [22]. Mealy machines are simpler than Moore machines with respect to the number of states, and so the models trained using this architecture are also simpler and more easily interpretable. We study the network’s capacity to solve several simple problems, including the recognition of regular languages, the computation of additions in different numerical bases and the generation of arithmetic expressions, and compare the results to standard RNNs with Elman architecture [8] and Long Short Term Memory (LSTM) networks [15]. We show that the Dual architecture can enhance the prediction accuracy and at the same time generate highly interpretable models.

The article is organized as follows. In section 2 we introduce the different network architectures used in all our experiments, including the Elman and LSTM networks that are used as a benchmark for comparison. In section 3 we describe the data and the experiments. Section 4 presents and analyzes the results. Finally, in section 5 we present the conclusions and discuss future lines of research.

2 NETWORKS

2.1 Elman RNN with noisy recurrence

The Elman RNN [8] is the simplest neural network architecture where recurrence is introduced. It adds a time dependence to the internal layer, making the activity in this layer depend on its output for the previous time step. Here we use a modified Elman RNN with one single hidden layer based on [22], where noise is introduced in
the activation function of the recurrent layer units. The network behavior is governed by the following equations:

\begin{align}
    h_t & = \tanh(W_{ch}x_t + W_{hh}h_{t-1} + X_v \circ h_{t-1} + b_h) \\
    y_t & = \sigma(W_{hy}h_t + b_y)
\end{align}

where \( h_t \) and \( y_t \) represent the activation of the hidden and output layers, respectively, at time \( t \), and \( x_t \) is the network input. The model depends on weight matrices \( W_{ch}, W_{hh} \) and \( W_{hy} \), and bias vectors \( b_h \) and \( b_y \). In particular \( W_{hh} \) represents the weights in the recurrent connection that makes a explicit dependence of \( h_t \) on \( h_{t-1} \). The recurrent connection also includes the noisy term \( X_v \circ h_{t-1} \), where \( X_v \) is a random vector whose elements are drawn from a normal distribution with mean \( 0 \) and standard deviation \( \nu \) each time the value of \( h_t \) needs to be computed. The \( \circ \) operator denotes an element-wise product. This noise is more effective for very active neurons, being negligible for silent ones. This way, the effect of the noise on the overall network’s behavior is to force the operation of neurons in an almost binary fashion [22].

\[ W_{sh}, W_{hc}, W_{xc} \]

\[ W_{hh} \]

\[ W_{hy} \]

\[ b_h \]

\[ b_y \]

\[ h_t \]

\[ y_t \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]

\[ h_{t-1} \]

\[ b_h \]

\[ b_y \]

\[ x_t \]

\[ X_v \]
and the expected output in this case is just the carry after the last two digits have been added.

We consider two different randomly generated datasets, each consisting of two input strings of length 200000 and the corresponding output string of the same length. The generation probability for the $\$ symbol is set to 0.1, and the rest of symbols are all equally probable. The first dataset is used to train the networks, the second is used for test. As before, the networks are evaluated by measuring their prediction accuracy on the test set.

Although the grammar associated to this problem is regular as in the previous case, the problem complexity increases with $B$ and the interpretation of a standard RNN solution becomes nontrivial. As we show in the results section, this kind of problem illustrates the benefits of using a network that separates the recurrent memory from the main processing path, such as the Dual RNN. Since the only information that needs to be remembered is the carry, the discharge of computing power in the recurrent layer represents a great advantage in this problem.

### 3.3 Generation of arithmetic expressions

Finally we test the models on a generation task. The networks are trained to predict the next symbol in an arithmetic expression that includes operators (+, -, *, /), parentheses and operands represented by the single symbol $a$, as well as the string separator $. Both the training and test datasets are single strings with 200000 symbols randomly generated according to the grammar rules:

$$S \rightarrow S\ op\ T | \ T$$
$$T \rightarrow a | (\ S\ )$$

#### Table 1: Description of the 7 Tomita Grammars

| Name      | Regular language                                                                 |
|-----------|-----------------------------------------------------------------------------------|
| Tomita1   | Strings with only $a$’s.                                                          |
| Tomita2   | Strings with only sequences of $ab$’s.                                            |
| Tomita3   | Strings with no odd number of consecutive $b$’s after an odd number of consecutive $a$’s. |
| Tomita4   | Strings with fewer than 3 consecutive $b$’s.                                      |
| Tomita5   | Strings with even length with an even number of $a$’s.                            |
| Tomita6   | Strings where the difference between the number of $a$’s and $b$’s is a multiple of 3. |
| Tomita7   | $b a^+b a^+$                                                                     |

and the expected output in this case is just the carry after the last two digits have been added.

We consider two different randomly generated datasets, each consisting of two input strings of length 200000 and the corresponding output string of the same length. The generation probability for the $\$ symbol is set to 0.1, and the rest of symbols are all equally probable. The first dataset is used to train the networks, the second is used for test. As before, the networks are evaluated by measuring their prediction accuracy on the test set.

Although the grammar associated to this problem is regular as in the previous case, the problem complexity increases with $B$ and the interpretation of a standard RNN solution becomes nontrivial. As we show in the results section, this kind of problem illustrates the benefits of using a network that separates the recurrent memory from the main processing path, such as the Dual RNN. Since the only information that needs to be remembered is the carry, the discharge of computing power in the recurrent layer represents a great advantage in this problem.

### 4 RESULTS

#### 4.1 Tomita Grammars

For the recognition of Tomita grammars only the noisy Elman and the Dual networks are considered. In both cases it is not difficult to find network parameters that allow for a perfect classification with a 100% accuracy in all tests. For the noisy Elman network we use 10 units in the recurrent layer, learning rate $r = 0.01$, L1 regularization $\lambda = 0.1$, a batch size of 10 and an unfold length of 25. The Dual network uses the same parameters, with 10 additional units in the feedforward layer. The networks are trained for 1000 epochs and we use an adaptive noise level that starts at $\nu = 0.0$ and linearly increases to reach a maximum value of $\nu = 1.0$ at epoch 500. We do not need to include the shocking mechanism reported in [22], maybe due to the use of adaptive noise.

After training, the recurrent layer is strongly regularized, with only a few neurons participating in the generation of the network’s output. Additionally, the active neurons operate in a binary regime, with their activity being pushed towards +1 or −1. No intermediate values are observed in the activation of recurrent units (see fig. 2).

#### Table 2: Example of inputs and output for the addition problem in base 2.

| Input 1     | $0101111010101010$          |
|-------------|-----------------------------|
| Input 2     | $1101011010101101$          |
| Output      | $010100110000110010010000$ |

#### Table 3: First 156 input characters from the training dataset in the arithmetic expression generation problem.

$$\$('((a)/a/a) + ((a-a*a)/(a-a-a+a))/a') + (a/(a-a-(a-a*a+a)))/a - (a-a-a+a)/(a-a-a+a)$

#### Table 4: Some examples of correct and incorrect strings generated by networks trained on the arithmetic expressions problem.

| Example                | Test | Why?    |
|------------------------|------|---------|
| $a - ((a + a) * /a + \(a\))$ | $\times$ | $+$    |
| $(a + (1) - a * ((a))$ | $\times$ | $(\)$ |
| $a - ((a) + a) / a + \(a\)$ | $\times$ | Depth -1 |
| $(((a)) + a * a / ((a))$ | $\times$ | Depth +1 |
| $(a + a) + a / (a/a)$ | $\sqrt{\}$ | |

The Networks are trained to predict the next symbol in an arithmetic expression that includes operators (+, -, *, /), parentheses and operands represented by the single symbol $a$, as well as the string separator $. Both the training and test datasets are single strings with 200000 symbols randomly generated according to the grammar rules.
Hence the activation patterns in the recurrent layer may be interpreted as a finite set of states, and state transitions in response to input symbols can be used to define a deterministic finite automaton (DFA) that summarizes the network behavior. This observation is general for all the networks that provide satisfactory test results, independently of their architecture.

It is however very interesting to compare the automata obtained for the Elman and the Dual architectures. Figure 3 shows these automata for the Tomita 6 problem after the application of a DFA minimization algorithm [16]. In the noisy Elman case the network’s output depends only on the recurrent layer state. This is represented by associating an output symbol to each automaton state. This way, the noisy Elman network accepts an interpretation as a Moore machine (figure 3, top). On the other hand, the Dual network’s output depends both on the recurrent layer state and the input. We may incorporate this information into the corresponding DFA by adding the output symbol to the transition labels. In this case the resulting automaton is a Mealy machine (figure 3, bottom).

Although the two automata seem in principle pretty similar, this observation is just valid for this particular problem. Note that for each state there exists only one valid transition with one single input symbol. Hence the Moore and Mealy machines have the same transitions graph. This panorama changes as soon as we consider more complex problems where the separation between input and memory represents a clear benefit in terms of interpretability. In those cases a network that admits an interpretation as a Mealy machine will be much simpler to understand. The addition problem considered next is one of such cases.

4.2 Addition

The addition problem described in section 3.2 has been approached using the noisy Elman and the Dual networks. The Elman RNN uses 20 units in the recurrent layer. The rest of parameters are as in section 4.1. As before, after proper training the network is able to correctly predict all the samples in the test set. Regularization and binarization are also observed, with only a few recurrent units coding the solution in the form of a finite set of states. Figure 4 shows the internal state space of one of such networks trained on the addition problem for $B = 2$. Only the three active units are shown (the rest are always silent). The three of them have a clear interpretation: the first neuron (N0) is learning the carry; the second (N2) keeps track of the carry in the previous time step; and the last one (N5) is dealing with the non-linearity of the binary addition problem, behaving as a XOR gate. A DFA can be extracted as before, and the network accepts again an interpretation as a Moore machine (see figure 5).

Figure 4: Activation plot of the three non-regularized neurons in an Elman RNN trained on the addition problem in base 2. For each neuron, the left plot represents the activation when there is no carry, while the right plot represents the activation when there is a carry from the previous step.
addition problem, since all the additional information needed to compute the state of N5 is contained in the current input. However, as the network has only one single recurrent layer concentrating all the processing power, some recurrent units are forced to learn information that does not explicitly depend on the past history. For more complex problems this could imply an unnecessary waste of memory resources, also hindering the interpretability.

This is the case for the addition problem when we consider higher bases. Figure 6 shows an example for $B = 10$. The network has 20 recurrent units and has been trained with the previous set of parameters. No errors are observed on the test dataset after training. As expected, only a few units survive the regularization and their activation is binary. However there is now only one neuron (N1) with a straightforward interpretation, and not surprisingly it is coding the carry. The other active neurons need to be used to compute the network’s output and, although their activation forms some characteristic patterns, their behavior is not meaningful at all. The extracted automaton is not shown because of its huge complexity.

Figure 5: Minimum DFA extracted from an Elman RNN trained on the addition problem in base 2.

A Dual network, with 10 neurons in the feedforward layer, has also been trained in the same conditions. After training, all the general observations extracted for the Elman case are still valid, but now only two neurons are active in the recurrent layer (figure 7). These two neurons are necessary to deal with the carry, and this is the main difference with respect to the Elman network: now the recurrent layer memorizes only what is strictly necessary to cope with time dependencies. The carry information stored in the recurrent layer, together with the network input, is sufficient for the additional feedforward layer to compute the correct output. This observation is valid regardless of the numerical base\(^2\).

The corresponding automaton is shown in figure 8. The result for the network trained with $B = 2$ has been used for the sake of clarity, but networks trained with different bases provide the same transition diagram (with additional labels for different input/output pairs). The automaton extracted from the Dual network is again a Mealy machine, but now the advantage over the Moore version associated to the Elman RNN is more evident (compare with the DFA shown in figure 5). As $B$ increases, the difference between the number of states needed by the recurrent layer in the two network architectures becomes more dramatic. While the Elman network needs more and more additional recurrent units to code the solution, the Dual RNN uses always the same memory configuration, leaving the main part of the computation to the feedforward layer. In summary, by allowing some of the processing be carried out by the feedforward layer, the Dual RNN discharges much of the computational load from the recurrent layer, letting it concentrate on just the information that must be remembered for future time steps. This is a very exciting result that could be of general application in more complex problems.

4.3 Arithmetic expressions

The last experiment tests the network capacity to generate arithmetic expressions with fixed parentheses depth. We consider Elman RNNs with 10, 20 and 30 units, Dual RNNs with 5, 10 and 15 units

\(^2\)We have tested with different $B$ values and in all the cases we obtain the same solution.
When L1 regularization is included, the accuracy decreases but it is with and without L1 regularization. The second half contains the noise. When noise is injected, however, the results consider-

_linear

The Dual RNN provides similar results when no noise is applied (not shown). When noise is injected, however, the results considerably improve (see table 6). With no regularization, all the considered networks reach more than 99% accuracy on the generation task. When L1 regularization is included, the accuracy decreases but it is still higher than for the Elman network. In all the cases the recurrent units get binarized and the activation patterns in the recurrent layer may be used to extract a DFA. As for previous problems, this automaton has the form of a Mealy machine. As an example, figure 9 shows the automaton extracted for the non-regularized 5-10 Dual network. In order to simplify the transitions graph, only the states with a parentheses depth lower than or equal to 2 are plotted and only transitions with a probability higher than 0.001 are represented. It is worth noting that these low probability transitions are responsible for the few grammatical errors observed in the generated strings, hence by discarding them we obtain an automaton that consistently generates the grammar.

Finally, to put these results in a proper context, we also consider the use of LSTMs on the same problem. The results are shown in table 7. The main observation is that, perhaps surprisingly, LSTMs are not able to achieve as high accuracy as the Dual networks on this prediction task. The best configuration provides an accuracy

| Config | Units | Test | Min | Max |
|--------|-------|------|-----|-----|
| noise = 0.0 | 5 - 10 | 99.2±0.6 | 97.9 | 99.8 |
| L1 = 0.0 | 5 - 10 | 99.3±0.5 | 98.1 | 99.9 |
| 10 - 10 | 99.3±0.5 | 97.9 | 99.8 |
| 10 - 20 | 99.3±0.6 | 97.9 | 100.0 |
| 15 - 10 | 99.6±0.3 | 99.0 | 99.9 |
| 15 - 20 | 99.7±0.1 | 99.5 | 99.8 |
| noise = 0.1 | 5 - 10 | 86.5±4.9 | 75.1 | 94.8 |
| L1 = 0.1 | 5 - 10 | 89.5±3.9 | 82.8 | 96.6 |
| 10 - 10 | 89.5±3.0 | 85.1 | 96.4 |
| 10 - 20 | 90.3±2.7 | 83.6 | 92.5 |
| 15 - 10 | 93.0±2.8 | 89.4 | 97.4 |
| 15 - 20 | 87.7±4.6 | 79.8 | 96.1 |

Table 5: Average accuracy of different Elman RNNs trained to generate arithmetic expressions.

Table 6: Average accuracy of different Dual RNNs trained to generate arithmetic expressions.
Figure 9: Minimum DFA extracted from a Dual RNN trained on the arithmetic expressions problem. Only the states with a parenthesis depth lower than or equal to 2 are plotted in order to simplify the graph. Transitions which are produced with a probability lower than 0.001 are also omitted.

Table 7: Average accuracy of different LSTM networks trained to generate arithmetic expressions.

| Config | Units | Test   | Min  | Max  |
|--------|-------|--------|------|------|
| L1 = 0.0 | 10    | 83.7±3.6 | 79.4 | 88.0 |
|        | 20    | 88.7±1.1 | 87.0 | 90.1 |
|        | 30    | 96.5±0.3 | 96.0 | 96.9 |
| L1 = 0.1 | 10    | 47.3±1.3 | 45.3 | 49.2 |
|        | 20    | 60.1±0.9 | 58.9 | 61.4 |
|        | 30    | 64.9±1.0 | 63.4 | 66.1 |

binarization. While the Dual networks can be easily understood as a DFA in Mealy form, the mechanisms used by LSTMs to solve the problem are not clear at all. On a different plane, the computational resources needed to train a Dual RNN are also lower in both memory and time.

5 CONCLUSIONS
In this article we have explored a neural network architecture that combines a recurrent layer connected to the network input and a feedforward layer that processes both the input and the output of the recurrent layer. This way, two different processing paths may focus on different aspects of learning. The recurrent path concentrates its resources on remembering information that must be preserved along time, so working as a kind of memory. The feedforward path is able to use this memory, together with the input, to provide the final network output. Networks using this architecture seem to make a better use of their computational resources, providing better results than traditional Elman RNNs and LSTM networks on prediction problems of different complexity. Additionally, the trained networks are more interpretable.

We have also observed that the introduction of noise in the activation function of the recurrent units forces these units to behave in a binary manner, with their output being always either +1 or −1. As a consequence, the time evolution of the network when it is presented a given input sequence can be seen as a transition through a finite set of discrete states. It is then possible to extract a transition map and show that the networks are internally behaving as deterministic finite automata. Although this observation is true also for the Elman RNNs, the automata extracted from networks that use the Dual architecture are much simpler, and implement a correct Mealy machine in all the problems we considered.

When facing a language generation problem, the Dual RNN also outperforms the Elman and LSTM architectures in terms of the grammatical correctness of the generated expressions, and the interpretability of the network as a Mealy machine is still preserved. In brief, we are able to train better networks that are in addition more interpretable. In spite of the simplicity of the considered language, we expect that this behavior can be extended to more complex problems, with potential applications in fields such as automatic music composition, natural language processing or machine translation. Even if the results on these areas did not achieve state of the art performance, the gain in interpretability might be worth the price.

ACKNOWLEDGMENTS
This work has been partially funded by grant S2017/BMD-3688 from Comunidad de Madrid and by Spanish project MINECO/FEDER TIN2017-84452-R (http://www.mineco.gob.es/).
