Evaluating the Learning of Automata through the Use of Recurrent Neural Networks

Lídio Mauro Lima de Campos, Alberto Sampaio Lima

***Abstract*— This paper presents the computational potentialities, demonstrating the learning capacity and extrapolation of recurrent neural networks (RNN) through the learning study of finite automata. Initially, the generalized delta rule is presented for a recurrent network. In addition, the theoretical basis on Finite Automata, Moore Machine and Mealy Machine are presented, showing the equivalence between them. We formalized the problem of parity using a Moore Machine and its equivalent Mealy Machine. We proceeded simulation and the results are presented for the problem and simulation of Tomita languages 1 and 2, testing the extrapolation capacity of each RNN. The results indicated that the neural networks with recurrent architectures have a good capacity of extrapolation for the simulated problems.**

***Index Terms*— Finite Automata, Applied computing, Recurrent Neural Networks.**

I. INTRODUÇÃO

Os avanços recentes na modelagem de redes neurais têm permitido grandes avanços em diversas áreas, tais como visão por computador, reconhecimento de voz [4], tradução automática [3], predição [5], tratamento de imagens [6], dentre outras. A capacidade de generalização de redes neurais pode ser subdividida nas capacidades de interpolação e extrapolação, por interpolação entende-se a capacidade de fornecer respostas corretas a questões que se situam dentro do intervalo de domínio das questões apresentadas durante a fase de aprendizado. A extrapolação é a capacidade de responder corretamente a questões que fazem parte do contexto, mas que estão fora do intervalo do domínio das questões vistas durante o aprendizado.

A extrapolação, portanto, é uma capacidade bem mais complexa e difícil que a interpolação. Acredita-se que um requisito para a extrapolação é aquisição de conhecimentos sobre os "princípios de funcionamento" do tema que envolve a questão que se quer resolver. As redes diretas com neurônios estáticos não são capazes de resolver estes princípios [2], principalmente quando os mesmos envolvem o funcionamento de sistemas dinâmicos.

Sabe-se que redes neurais diretas podem facilmente resolver o problema da paridade para um vetor de entrada com um certo número de *bits*.

Lídio Mauro Lima de Campos, Universidade Federal do Pará (UFPA), Castanhal, Brasil, limadecampos@gmail.com

Alberto S. Lima, Universidade Federal de Campina Grande (UFCG), Campina Grande, Brasil, albertosampaio@ufc.br

Por exemplo, se fosse apresentado na entrada um vetor de 8 *bits*, uma rede com 8 neurônios na camada de entrada, 8 neurônios na camada escondida e 1 neurônio na camada de saída e uma amostra dos 256 possíveis vetores de entrada como conjunto de treinamento, seria suficiente para resolver o problema. Esta rede deveria ser capaz de essencialmente aprender a contar o número de *bits* 1 no conjunto de treinamento.

Além disso, após o treinamento, a rede neural direta seria capaz de interpolar entre os exemplos apresentados, chegando a acertar o valor da paridade para todos os 256 valores possíveis de serem questionados. Contudo a rede direta não seria capaz de responder qual a paridade para um vetor de entrada de 9,10, ou n *bits*, ou seja, a rede não é capaz de extrapolar. Vê-se que a própria topologia limita a sua capacidade de extrapolação. Uma maneira de fazer com que a rede fosse capaz de entender o "princípio de funcionamento" da questão da paridade, seria ensinar para ela, por exemplo, o próprio autômato que determina a paridade. Para implementar o problema da paridade, portanto, necessita-se de uma rede com topologia recorrente.

As redes neurais recorrentes (RNRs) são aproximadores universais de sistemas dinâmicos [7]. Por meio do algoritmo de retropropação com atraso no tempo (*backpropagation through time*), uma rede recorrente pode aprender pesos que lhe permitem armazenar memórias de curto prazo em sua dinâmica. A RNR relaciona temporalmente, conforme necessário, eventos separados para realizar classificações ou predições de séries temporais. Portanto, nessa pesquisa se testa a capacidade de extrapolação redes neurais recorrentes, através do aprendizado de diversos autômatos e linguagens formais.

Na seção 2 é apresentada a fundamentação teórica que norteou a pesquisa, bem como os trabalhos relacionados. Na seção 3 apresenta-se o estudo realizado, com a análise dos resultados. Finalmente na seção 4 apresentam-se as conclusões da pesquisa.

II. FUNDAMENTAÇÃO TEÓRICA E TRABALHOS RELACIONADOS

Uma rede neural direta, como ilustrado na Fig. 1, consiste em várias camadas compostas por nós (neurônios) onde cada neurônio de uma camada possui ligação com todos os neurônios da camada seguinte. Conforme visto na Fig. 1, a rede direta, denominada perceptron multicamada, possui uma camada de entrada, que atua como os sensores da rede captando os estímulos do ambiente e pode ter uma ou mais camadas intermediárias que é onde a maior parte do processamento é realizada através das conexões e seus pesos respectivos, podem ser considerados como extratoras de

características e uma camada de saída onde o resultado final é concluído e apresentado. Ainda na Fig. 1 não existem conexões entre a saída de um neurônio e algum outro neurônio localizado em uma camada anterior ao primeiro, ou seja, não possui ciclos, fato que caracteriza uma rede *feedforward*.

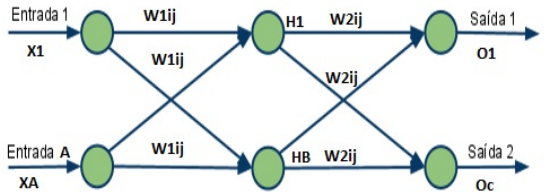


Fig. 1. Rede Perceptron Multicamada.

Redes em ciclo (ou com realimentação, ou com retroação, ou com "realimentação") são aquelas cujo grafo de conectividade contém ao menos um ciclo, a Fig. 2 ilustra uma rede recorrente com a saída realimentada. Quando, além disso, envolve neurônios dinâmicos são chamadas recorrentes. As realimentações envolvem a utilização de ramos particulares compostos por atrasos unitários denotados por z^{-1}, o que resulta em um comportamento dinâmico não-linear em virtude da natureza não-linear dos próprios neurônios. Esta característica de dinâmica não-linear da estrutura permite que este tipo de rede neural seja utilizado em problemas e aplicações que necessitem representar estados, tais como: predição de consumo de energia, processamento de voz, controle industrial, processamento adaptativo de sinais e predição de séries temporais. Hoje em dia as redes neurais recorrentes têm sido objeto de extensa pesquisa e num futuro próximo, acredita-se, suplantará em utilização as redes neurais diretas, justamente pela capacidade de representar sistemas dinâmicos, inexistentes nas redes diretas com neurônios estáticos.

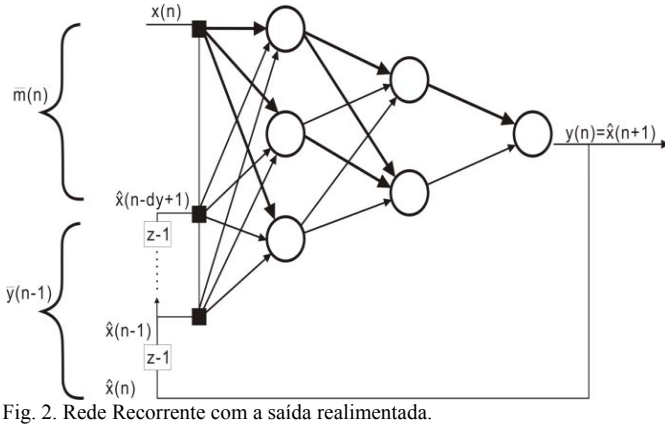


Fig. 2. Rede Recorrente com a saída realimentada.

O algoritmo de treinamento utilizado para a atualização dos pesos da RNR mostrada na Fig. 2, considerando os atrazos no tempo, foi *Backpropagation Through Time-(BPTT)* (DONG; KAISHENG; YU, 2015). A formulação matemática do mesmo é descrita a seguir, utilizando a seguinte notação: Denota-se os níveis de ativação das unidades da camada de entrada por xj, da camada oculta por hj e da camada de saída por oj. A matriz de pesos da camada de entrada para a intermediária é denotada por ***V***, enquanto que a da camada oculta para a de saída é denotada por ***W***. A Camada de entrada consiste em dois componentes, **x(t)** e a ativação anterior da camada oculta **o(t -1)**. A matriz de pesos correspondentes é dada por ***U***. Onde o erro na camada de saída é denotado por:

$$e_o^{(t)} = o_j^{(t)} - y_j^{(t)} \tag{1}$$

A matriz de atualização de pesos da camada de saída é atualizada de acordo com a equação 2:

$$W_{(t+1)} = W_{(t)} + \eta\, h_{(t)} e_{0(t)}^T \tag{2}$$

Em seguida, os gradientes de erros são propagados da camada de saída para a camada oculta:

$$e_h^{(t)} = d_h\ (e_{0(t)}^T V, t) \tag{3}$$

Em que o vetor de erro é obtido utilizando a função de $d_h\,(\cdot)$

$$d_{h,t}(x,t) = xf^{'}(net_j) \tag{4}$$

A matriz de pesos **V** entre a camada de entrada e camada oculta é atualizada conforme:

$$V_{(t+1)} = V_{(t)} + \eta\, x_{(t)} e_{n(t)}^T \tag{5}$$

A matriz de pesos recorrentes é atualizada por:

$$U_{(t+1)} = U_{(t)} + \eta\, o_{(t-1)} e_{n(t)}^T \tag{6}$$

De acordo com [10], o aprendizado profundo (*deep learning - DL*) em redes neurais é relevante para o aprendizado supervisionado, aprendizado não supervisionado e aprendizado de reforço. Aliviando problemas com caminhos profundos de atribuição de crédito, o aprendizado não supervisionado não só pode facilitar o aprendizado supervisionado de sequências e padrões estacionários, mas também o aprendizado de reforço. Os autores afirmam que a programação dinâmica é importante para o aprendizado supervisionado profundo e aprendizado de reforço tradicional em redes neurais.

Em [11], os autores apresentaram um controlador baseado em RNRs, parte de um sistema que aprende, o qual acessava o ambiente a partir de uma série de *interfaces* pré-definidas, selecionadas manualmente de acordo com o tipo de tarefa. A pesquisa explorou a capacidade dos modelos de rede neural no intuito de aprender algoritmos para operações aritméticas simples. Através de experimentos com supervisão e reforço de aprendizagem, foi mostrado que as RNRs são capazes desse aprendizado com sucesso, embora com algumas ressalvas, já qie os autores não encontraram um controlador único que pudesse resolver todas as tarefas.

O trabalho apresentado em [12] apresenta uma máquina de estado finito (FSM) que visa reduzir o tempo de espera do usuário para obter o resultado do reconhecimento de

manuscritos em inglês. Os resultados indicaram uma redução no tempo de espera para a decodificação de mais de 56% no IAM-OnDB e de 63% em IBM_UB_1 sem taxa de reconhecimento degradante.

As redes neurais recorrentes são modelos cada vez mais populares para dados seqüenciais. A arquitetura da rede neural recorrente é simples, no entanto, não é adequada para capturar dependências de longa distância. Em sua pesquisa [13], os autores buscaram aumentar a eficiência de representação, ou seja, a proporção de desempenho para os parâmetros adquiridos. A partir do *framework* RNR, a arquitetura do modelo foi simplificada e foram desenvolvidas várias variantes, onde o estado da camada oculta do próximo passo é uma função do seu estado atual e da mudança delta calculada pelo modelo, em um *framework* denominado RNR Delta, que obteve um melhor desempenho em tarefas de modelagem de linguagem, sendo considerado conceitualmente muito mais simples e com muito menos parâmetros [13].

As redes neurais tem sido utilizadas nas mais diversas aplicações. Em [15] foram comparadas as capacidades das Redes Neurais Artificiais (RNA) e do Planejamento Experimental Fatorial (FEP) para medir as variáveis mais significativas no processo de moagem que influenciam a força de corte.

A. Autômatos finitos

Um autômato finito pode ser visto como uma máquina composta, basicamente, de três partes:
a)**Fita** - Dispositivo de entrada que contém a informação a ser processada.
b)**Unidade de Controle** – Reflete o estado corrente da máquina, possui uma unidade de leitura (cabeça da fita), a qual acessa uma fita cada vez e movimenta-se exclusivamente para a direita.
c)**Programa ou Função de transição** – Função que comanda as leituras e define o estado da máquina.

Definição 1: Um autômato finito determinístico ou simplesmente autômato finito é uma 5-upla M=(Σ,Q, δ,qo,F) onde:

Σ é o alfabeto de símbolos de entrada, Q conjunto de estados possíveis do autômato o qual é finito, Σ função programa ou função de transição **: Q x Σ→Q,** **qo** estado inicial tal que **qo** é elemento de Q, **F** conjunto de estados finais tal que **F** está contido em **Q**.

Autômatos Finitos com Saída

O conceito básico de Autômato Finito (AF) possui aplicações restritas, pois a informação de saída é limitada à lógica binária aceita/rejeita. Sem alterar a classe de linguagens reconhecidas. É possível estender a definição de AF incluindo a geração de uma palavra de saída. As saídas podem ser associadas às transições (Máquina de Mealy) ou aos estados (Máquina de Moore). Em ambos as máquinas, a saída não pode ser lida, ou seja, não pode ser usada como memória auxiliar, e é como segue:
•É definido sobre um alfabeto especial, denominado alfabeto de saída (pode ser igual ao alfabeto de entrada);
•A saída é armazenada em uma fita independente da de entrada;
•A cabeça da fita de saída move uma célula para direita a cada símbolo gravado;
•O resultado do processamento do autômato finito é o seu estado final (condição de aceita/rejeita) e a informação contida na fita de saída.

Máquina de Mealy

A máquina de Mealy é um Autômato finito modificado de forma a gerar uma palavra de saída para cada transição.

Definição 2: Uma máquina de Mealy M´ é um autômato finito determinístico com as saídas associadas às transições. É representada por uma 6-upla: $\mathbf{M'=(\Sigma,Q,\delta,qo,F,\Delta)}$ onde:
Σ é o alfabeto de símbolos de entrada, **Q** conjunto de estados possíveis do autômato o qual é finito, função programa ou função de transição **δ : Q x Σ→QxΔ*** , **qo** estado inicial tal que qo é elemento de Q, **F** conjunto de estados finais tal que F está contido em Q, **Δ** Alfabeto de símbolos de saída.

Máquina de Moore

A máquina de Moore possui uma segunda função, que gera uma palavra de saída (que pode ser vazia) para cada estado da máquina.

Definição 3: Uma máquina de Moore M´´ é um autômato finito determinístico com as saídas associadas aos estados. É representada por uma 7-upla: M´´= **(Σ,Q,δ,qo,F,Δ, δs)** onde:
Σ é o alfabeto de símbolos de entrada, **Q** conjunto de estados possíveis do autômato o qual é finito, **δ** função programa ou função de transição, **δ : Q x Σ→Q, qo** estado inicial tal que qo é elemento de Q, **F** conjunto de estados finais tal que **F** está contido em **Q**, **Δ** Alfabeto de símbolos de saída, **δs** função de saída : **δs: Q→Δ***.

Equivalência das Máquinas de Mealy e Moore

Teorema 1: Toda máquina de Moore pode ser simulada por uma Máquina de Mealy, para entradas não vazias.
Prova: Suponha **MO=(Σ, Q, δMO, qo, F, Δ, δs)**, uma Máquina de Moore qualquer. Seja: uma Máquina de Moore qualquer. Seja: **ME=(Σ, Q∪{qe},δME,qe,F,Δ)** uma máquina de Mealy onde a função **δME** é definida como segue (suponha q um estado de **Q** e a um símbolo de **Σ**):

$$\delta ME(qe,a)=(\delta MO(qo,a),\delta s(qo)\ \delta s((\delta MO(qo,a))) \quad (7)$$

$$\delta ME(q,a)=(\delta MO(q,a),\ \delta s((\delta MO(q,a))) \quad (8)$$

Na equação (8) é construída a função programa da Máquina de Mealy, a partir das funções de transição e de saída

da Máquina de Moore. O estado **qe** introduzido na equação (7) é referenciado somente na primeira transição a ser executada. Seu objetivo é garantir a geração da saída referente ao estado inicial **qo** de Moore. Um exemplo de simulação de Máquina de Moore utilizando Máquina de Mealy (Fig. 3). Pela definição de máquina de Moore, tem-se:
MO=(**Σ, Q, δMO, qo, F, Δ, δs**) **onde: Σ={ao,a1}, Q ={qo,q1}, δMO: QxΣ→Q, qo , F={qo,q1}, Δ={uo,u1}, δs= Q→Δ*.**

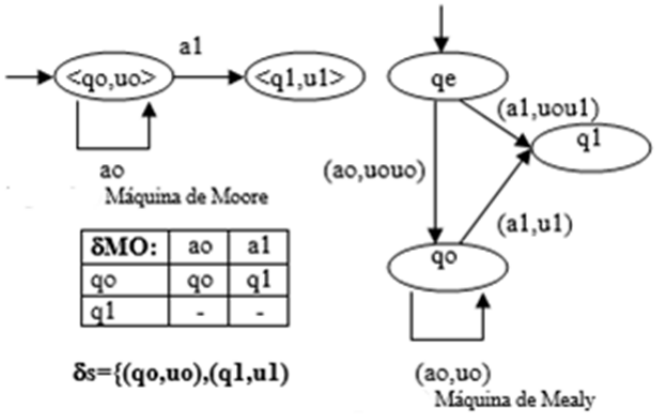


Fig. 3. Máquinas de Moore e Mealy [14].

A obtenção da Função Programa Máquina de Mealy, baseada na Função programa e de Saída da Máquina de Moore é mostrada na sequência.

δME(qe,ao)=(δMO(qo,ao), δs(qo) δs((δMO(qo,ao))) , **δME(qe,ao)=(qo,uouo)**

δME(qe,a1)=(δMO(qo,a1), δs(qo) δs((δMO(qo,a1))) , **δME(qe,a1)=(q1,uou1)**

δME(qo,ao)=(δMO(qo,ao), δs(qo) δs((δMO(qo,ao))), **δME(qo,ao)=(qo, δs(qo))=(qo,uo)**

δME(qo,a1)=(δMO(qo,a1), δs(qo) δs((δMO(qo,a1))), **δME(qo,a1)=(q1, δs(q1))=(q1,u1)**

Equivalência das Máquinas de Moore e Mealy

Teorema2: Toda máquina de Mealy pode ser simulada por uma máquina de Moore.

Prova: Suponha **ME=(Σ, Q, δME, qo, F, Δ)**, uma Máquina de Mealy qualquer. A máquina de Moore pode ser construída a partir de **δME** onde a função **δMO** é como segue:

- **Condição 1**: Para **a** em **Σ,** se **δME(qo,a)=(q,u)** então: **δMO=(<qo,ε>,a)=<q, u>**
- **Condição 2:** Para **b** em **Σ**, e para **q** em **Q**,se **δME(q,b)=(p,v),** então, para **ai** em **Σ** e para **qi** em **Q** tais que **δME(qi,ai)=(q,ui)**, tem-se que: **δMO=(<qo,ui>,b)=<p, v>** e onde a função de saída **δs** é tal que, para o estado **<q,u>** de **MO**: **δs=(<q,u>)=u**

Como exemplo considere uma Máquina de Mealy **ME=({a,β},{q,p},δME,q,{q,p}, {a,β}**}, simulando a Máquina de Moore, que compacta os brancos de um texto onde a representa um símbolo qualquer do texto e **β** o símbolo branco, como ilustrado na Fig. 4 (na etiqueta de uma transição, a primeira componente representa o símbolo lido e a segunda a palavra gravada). A máquina de Moore construída conforme o Teorema 2 é **MO==({a,β},Q,δMO,<q,ε>,F,{a,β},δs})**, tal que **Q=F={q,p}x{ε,a,β}**, ilustrada na Fig. 5, onde a segunda componente de cada estado representa a saída.

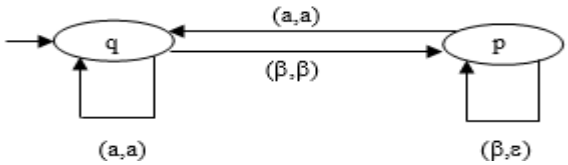


Fig. 4. Máquina de Mealy [14].

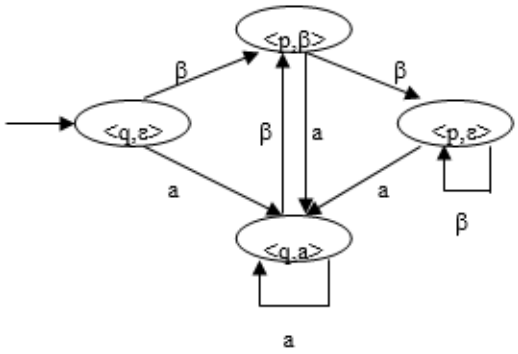


Fig. 5. Máquina de Moore [14].

A obtenção da função programa e da função saída da máquina de Moore é obtida pela condição 1 e condição 2. Inicialmente, aplicando-se a condição 1, tem-se:

- Para **a** em **Σ**, se **δME(qo,a)=(q,a);** então **δMO(<q,ε>,a)=<q,a>** e **δs(<q,ε>)=ε.**
- Para **β** em **Σ**, se **δME(qo, β)=(p,β);** então **δMO(<q,ε>,β)=<p,β>** e **δs(<q,ε>)=ε.**
- Usando-se a condição 2, obtém-se
- Para **a** em **Σ**; **q** em **Q;** se **δME(q,a)=(q,a)**; então **β** em **Σ** e para **p** em **Q**, tais que: **δME(p, β)=(p, ε)**, tem-se que **δMO(<p,ε>,a)=<q,a>** e **δs(<p,ε>)=ε.**
- Para **a** em **Σ**; **p** em **Q**; se **δME(p,a)=(q,a)**;então **β** em **Σ** e para **q** em **Q**, tais que:**δME(q, β)=(p, β)**,tem-se que:**δMO(<p,β >,a)=<q,a>** e **δs(<p,β >)=β.**
- Para **β** em **Σ**; **q** em **Q**; se **δME(q, β)=(p, β),** então **a** em **Σ** e para **p** em **Q**, tais que:**δME(p, a)=(q, a)**,tem-se que:**δMO(<q,a >,β)=<p, β>** e **δs(<q,a>)=a**.
- Para **a** em **Σ**; **q** em **Q**; se **δME(q, a)=(q, a)**, então **a** em **Σ** e para **p** em **Q,** tais que:**δME(p, a)=(q, a)**,tem-se que:**δMO(<q,a >,a)=<q, a>** e **δs(<q,a >)=a.**
- Para **β** em **Σ**; **p** em **Q;** se **δME(p, β)=(p, ε),** então **β** em **Σ** e para **q** em **Q,** tais que:**δME(q, β)=(p, β),**tem-se que:**δMO(<p, β >,β)=<p, ε>** e **δs(<p,β >)=β.**

- Para **β em Σ; p em Q**; se **δME(p, β)=(p, ε),** então **β** em **Σ** e para **p** em **Q**, tais que**:δME(p, β)=(p,ε)**,tem-se que**:δMO(<p, ε>,β)=<p, ε>** e **δs(<p,ε >)=ε.**

Classes de Problemas

Minsky e Papert [1] definiram a ordem de um predicado como sendo o tamanho da maior conjunção na forma lógica da mínima soma-de-produtos para aquele predicado, ou seja, quantas linhas retas são necessárias para separar os pontos que compõem as categorias do predicado. Assim, enquanto "OU" e o "E" são predicados de primeira ordem, "ou-exclusivo" é um predicado de ordem **n**.

Após catalogar a ordem de várias funções geométricas. Minsky e Papert [1] voltaram sua atenção para o problema do aprendizado. Eles mostraram que à medida que a ordem dos predicados aumenta, o número de coeficientes pode crescer exponencialmente levando a um sistema a necessitar quantidades enormes de memória e estendendo o processo de convergência a um número infinito de iterações.

Pode-se resumir a idéia de Minsky e Papert [1], afirmando-se que redes neurais diretas com neurônios estáticos são capazes de aprender apenas funções lógicas de primeira ordem, ou seja, distinguir apenas padrões linearmente separáveis ou predicados de ordem 1.

Já redes neurais diretas com neurônios estáticos com uma camada intermediária de neurônios e função de saída não-linear são capazes de modelar qualquer função estática no grau de precisão que desejar, ou seja, estas redes podem resolver predicados de ordem 2 ou superior, no entanto, o número de coeficientes necessários para resolver predicados de ordem muito alta pode tornar inviável a utilização deste tipo de rede.

Após se chegar nos problemas dinâmicos, vê-se que apenas redes neurais recorrentes são capazes de resolver este tipo de problema. A questão agora, diz respeito à topologia da rede recorrente e que tipo de problemas dinâmicos ele é capaz de resolver.

B. A paridade como um problema de ordem infinita

O problema da paridade, consiste em apresentar na entrada da rede neural uma "string" de *bits* e obter na sua saída o valor da paridade da "string" apresentado. O grafo correspondente a Máquina de Mealy, que implementa o problema da paridade, é mostrado na Fig. 6), onde a segunda componente de cada estado representa a saída, ou seja, o bit que indica a paridade de uma string de bits. Analogamente, pode-se obter a Máquina de Moore que simula a Máquina de Mealy da Fig. 6 e que também implementa o problema da paridade. O grafo correspondente, que implementa o problema da paridade é mostrado na Fig. 7), onde a segunda componente de cada transição, representa a saída, ou seja, o bit que indica a paridade de uma "string" de *bits.*

A inserção do novo estado <0, ε> é necessário, pois, senão antes de começar o processamento, no estado inicial, a máquina estaria escrevendo 0 na saída. Com a introdução deste estado, ela continua escrevendo, só que, desta vez, a cadeia vazia. A saída 0 significa paridade par e a saída 1 corresponde à paridade impar. Assim se o autômato estava no estado O e o próximo bit da "string" for 1, a paridade passa a ser impar (saída 1), caso o próximo bit fosse 0, a paridade permaneceria sendo par (saída 0). Se o autômato estivesse no estado 1 e o próximo bit da "string" for 1, a paridade passa a ser par (saída 0), caso o próximo bit fosse 0, a paridade permaneceria sendo impar (saída 1).

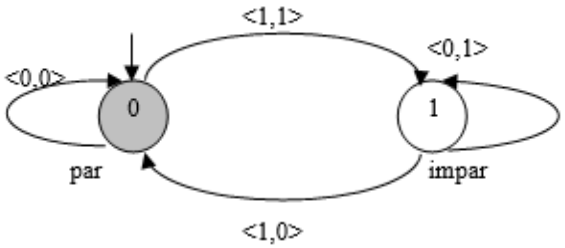


Fig. 6. Grafo da correspondente Máquina de Mealy
$ME = \{\sum, Q, \partial, qo, F, \Delta\} = (\{0,1\},\{0,1\},\delta,0,\{0,1\},\{0,1\})$,
que implementa o cálculo de paridade de uma string

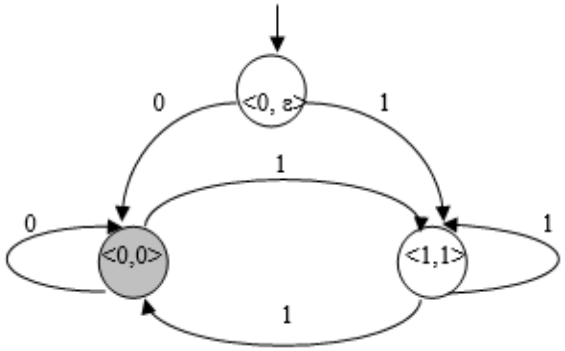


Fig. 7. Grafo da correspondente Máquina de Moore
$MO = \{\sum, Q, \partial, qo, F, \Delta, \delta s\} = (\{0,1\},\{0,1\},\delta,0,\{0,1\},\{0,1\},\delta s)$
que implementa o cálculo de paridade de uma string

Uma maneira de fazer com que uma rede neural fosse capaz de entender o "princípio de funcionamento" da questão da paridade seria ensinar para ela, por exemplo, o próprio autômato (Máquina de Mealy ou Moore) que determina a paridade, da Fig. 6 ou Fig. 7. Para implementar o autômato considere uma rede recorrente com apenas uma camada de neurônio de entrada e uma camada de neurônios de saída.

A saída de cada neurônio é ligada através de um conjunto de linhas de retardo à entrada de cada um dos outros neurônios da camada de saída, inclusive para si próprio, como pode ser visto na Fig. 8.

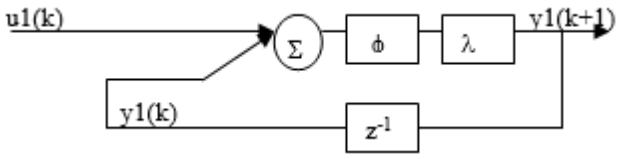


Fig. 8. Rede Neural Artificial Recorrente sem camada intermediária

A rede recorrente contém apenas um neurônio de entrada e um neurônio de saída e pode ser visto na Fig. 8. Uma maneira de treinar esta rede é utilizar o algoritmo do tipo

backpropagation. Para isto, durante a fase de treinamento a rede é transformada em uma rede direta equivalente e o conjunto de vetores de entrada para o treinamento inclui o valor do estado anterior do autômato finito.

A rede direta equivalente pode ser vista na Fig. 9 e o conjunto de treinamento na Tabela 1.

TABELA I
CONJUNTO DE TREINAMENTO PARA O AUTÔMATO QUE IMPLEMENTA O CÁLCULO DA PARIDADE

Entrada u(k)	*Estado x(k)*	*Estado x(k+1)*
0	*0*	*0*
1	*0*	*1*
0	*1*	*1*
1	*1*	*0*

Ao analisar a Tabela I, contendo o conjunto de treinamento, percebe-se claramente que o conjunto de treinamento representa a função lógica "OU-EXCLUSIVO".

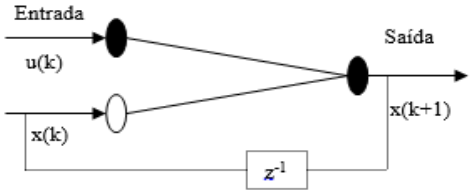


Fig. 9. Rede Neural Direta equivalente à rede recorrente sem camada intermediária incapaz de aprender o autômato finito da paridade

Como a rede neural direta equivalente à rede recorrente para o treinamento não possui camada intermediária de neurônios, é impossível implementar esta função com esta topologia de rede neural. Logo autômatos que implementam extensões de predicados de ordem maior que 1, não podem ser implementados utilizando redes neurais recorrentes sem camada intermediária de neurônios. Por extensão percebe-se que redes recorrentes com uma camada intermediária são capazes de implementar qualquer autômato finito. Uma rede idêntica à mostrada na Fig. 2, com um neurônio na camada de entrada, um valor diferente de zero neurônios na camada escondida e um neurônio na camada de saída, pode implementar o autômato.

C. Redes de Tomita

Um tópico de pesquisa que tem despertado o interesse de pesquisadores na área conexionista consiste no desenvolvimento de modelos de RNA com a capacidade de induzir autômatos finitos baseados apenas em exemplos positivos e negativos de "strings" ou cadeias, que pertencem e que não pertencem à linguagem reconhecida pelo mesmo. Uma coleção explícita de exemplos positivos e negativos, mostrados na Tabela II, e que colocam dificuldades específicas para a indução da linguagem pretendida é apresentada em (TOMITA, 1982).

A caracterização das linguagens propostas é apresentada na Tabela III. Percebe-se que os conjuntos de treinamentos não são balanceados, ou seja, são incompletos e variam enormemente na sua capacidade de definir a linguagem pretendida.

TABELA II
EXEMPLOS POSITIVOS E NEGATIVOS DE ALGUMAS LINGUAGENS INVESTIGADAS POR TOMITA

Linguagem	**Exemplos Positivos**	**Exemplos Negativos**
1	ε,1,11,111,1111,11111,111111, 1111111,11111111	0,10,01,00,011,110,000, 11111110,10111111
2	ε,10,1010,101010, 10101010,10101010101010	1,0,11,00,101,100 10001010,10110,110101010
3	ε,1,0,01,11,00,100,110,111,000, 100100,110000011100001, 111101100010011100	10,101,010,1010,1011,111010, 1001000,1111100, 0111001101,11011100110
4	ε,1,0,10,01,00,100100, 001111110100,0100100100,11100,01 0	000,11000,0001,000000000,0000 0 ,0000,11111000011, 1101010000010111,1010010001
5	ε,10,01,1100,111,000000 0111101111,100100100	1,0,11,00,101,011,11001,1111, 00000000,0101111,10111101111, 1001001001
6	ε,1,0,10,01,11111,000,00110011, 0101,00001000011111,00100, 011111011111,00	1010,00110011000,0101010101, 1011010,10101,010100,101001, 100100110101

TABELA III
CARACTERIZAÇÃO DE ALGUMAS LINGUAGENS PROPOSTAS POR TOMITA

Linguagem	**Descrição**
1	1*
2	(10)*
3	Nenhuma cadeia com número ímpar de 0´s é permitido depois de uma cadeia com números ímpar de 1´s.
4	Não mais do que dois 0´s em uma cadeia
5	(número de 1´s-número de 0´s) mod 3=0
6	0*1*0*1*

III. ESTUDO DE CASO E ANÁLISE DOS RESULTADOS

Para satisfazer os objetivos da presente pesquisa, foi realizado um estudo de caso [8, 9], a partir de simulações realizadas em diferentes cenários.

Simulações - Paridade

As simulações foram realizadas em duas etapas: treinamento e operação da rede. A primeira etapa teve por objetivo de ensinar o autômato da Fig. 6 para a rede. O conjunto de treinamento apresentado foi a string de bits 0,1,0,1,1,0,1 na entrada e na saída a string 0,1,1,0,1,1,0.

Utilizou-se uma rede de Recorrente (vide Fig. 2), com um neurônio na camada de entrada, cinco na camada intermediária e um neurônio na camada de saída, o erro admitido foi de 0.0001 e o número de épocas para o treinamento foi de 30000. Uma vez que a rede aprendeu o conceito de o autômato, o próprio conceito de paridade foi aprendido, com isso a rede será capaz de calcular a paridade de um vetor com um número qualquer de bits, bastando que os bits sejam apresentados sequencialmente ao neurônio de entrada da rede.

A segunda etapa teve por objetivo testar o aprendizado da rede e comprovar a capacidade de extrapolação das redes recorrentes, os resultados são mostrados nas Tabelas IV, V e VI. Para a "string" de *bits* 0,1,0,1,1,0,1 apresentada na entrada da rede a saída desejada (ou sequência de estados) seria 0,1,1,0,1,1,0, o que equivale a paridade par, pois a última saída é 0. As saídas obtidas e os respectivos erros são mostrados abaixo na Tabela IV. O segundo experimento considerou como entrada a string 0,0,1,0,1,0,1 e saída desejada a string 0,0,1,1,0,0,1, o que representa paridade impar, pois a última saída é 1, o que é mostrado na Tabela 5. O terceiro experimento, apresentou-se a seguinte string de bits na entrada da rede 0,0,1,0,1,0,1,1,0,1,0 e saída desejada a string 0,0,1,1,0,0,1,0,0,1,1, ou seja, uma string que tem paridade impar, os resultados são mostrados na Tabela VI.

TABELA IV
SAÍDAS OBTIDAS E DESEJADAS PARA A ENTRADA 0,1,0,1,1,0,1

Saídas Desejadas	Saídas Obtidas	Erro
0	0.022707	0.000258
1	0.987723	0.000121
1	0.987723	7.5357E-05
0	0.016073	0.000129
1	0.984752	0.000116
1	0.987743	7.511157e-05
0	0.016074	0.000129

TABELA V
SAÍDAS OBTIDAS E DESEJADAS PARA A ENTRADA 0,0,1,0,1,0,1

Saídas Desejadas	Saídas Obtidas	Erro
0	0.022707	0.000258
0	0.002135	0.000228
1	0.984533	0.00012
1	0.987728	7.530615E-5
0	0.016074	0.000129
0	0.02168	0.000235
1	0.984519	0.00012

TABELA VI
SAÍDAS OBTIDAS E DESEJADAS PARA A ENTRADA 0,0,1,0,1,0,1,1,0,1,0

Saídas Desejadas	Saídas Obtidas	Erro
0	0.013282	8,020929E-05
0	0.01286	8.2692223E-05
1	0.990632	4.387609E-05
1	0.992627	2.718277E-05
0	0.009866	4.867278E-05
0	0.012957	8.393564E-05
1	0.99063	4.389973E-05
0	0.009821	4.82228E-05
0	0.012958	8.395306E-05
1	0.99063	4.390006-05
1	0.992627	2.718361E-05

Simulações - linguagem de Tomita

Nas simulações das linguagens de Tomita, os valores adotados para as entradas da rede neural foram (0.5,1,0) respectivamente para os valores (0,1,ε) e (1,0) para as saídas correspondendo a exemplos (+,-). A seguir apresentam-se os resultados obtidos por simulação para as linguagens de Tomita 1 e 2, bem como os parâmetros utilizados e testes da capacidade de extrapolação. O valor do erro aceitável foi de 10^{-2} e o número de épocas usadas foi de 30000. Finalmente nas Tabelas VII e VIII, mostram-se os testes de operação das redes para as duas linguagens.

TABELA VII
OPERAÇÃO DA REDE PARA O PROBLEMA LIGUAGEM 1 DE TOMITA

x(k+1) Obtido	x(k+1) Desejado	|Erro|
0.993624	1	0.006376
1	1	0
1	1	0
1	1	0
0.00578	0	0.00578
0.026471	0	0.026471
0.978195	1	0.021805
1	1	0
0.00578	0	0.00578
0.026471	0	0.026471
0.005959	0	0.005959

TABELA VIII
LISTA DE IMPACTO SIMULTÂNEA

x(k+1) Obtido	x(k+1) Desejado	|Erro|
0.760134	1	0.239866
0.092165	0	0.092165
0.873081	1	0.126919
0.092139	0	0.092139
0.873109	1	0.126891
0.092139	0	0.092139
0.873109	1	0.126891
0.092139	0	0.092139
0.873109	1	0.126891
0.098142	0	0.098142
0.096041	0	0.096041
0.868848	1	0.131152
0.096201	0	0.096201
0.86867	1	0.13133
0.09214	0	0.09214
0.873109	1	0.126891

IV. CONCLUSÃO

O presente trabalho apresentou um estudo sobre a utilização de redes neurais recorrentes no aprendizado de autômatos.

A partir da análise dos resultados obtidos nas simulações realizadas, pode-se concluir que o aprendizado em redes neurais recorrentes é mais complicado do que o aprendizado em redes diretas. A escolha adequada do conjunto de treinamento também é de suma importância para sucesso no processo.

O uso de heurísticas, tais como modificação da taxa de aprendizado, aumento ou diminuição de neurônios na camada intermediária, facilitam também a convergência da rede. Pelos resultados mostrados pode-se perceber que a rede é capaz de calcular a paridade de um número ilimitado de *bits* de entrada.

Esse cálculo não seria possível com a utilização de uma rede direta, pois se a entrada da rede tivesse **n** *bits,* uma rede com **n** neurônios na camada de entrada, **n** na camada escondida, 1 neurônio da camada de saída e uma amostra representativa de **2n** possíveis vetores de entrada como conjunto de treinamento, seria possível resolver o problema.

Essa rede deveria essencialmente aprender a contar o número de *bits* no conjunto de treinamento. Após a rede treinada a rede seria capaz de interpolar o exemplo apresentado, chegando a acertar o valor da paridade para todos os **2n** valores possíveis de serem questionados. Entretanto ela não é capaz de resolver qual a paridade para um vetor de entrada de maior que **n** *bits*. O que é possível com a utilização de redes recorrentes que possuem uma excelente capacidade de generalizar no sentido de extrapolação. De acordo com os resultados promissores de simulações apresentados nas Tabelas IV, V, VI, VII, VIII percebe-se que as RNAs com arquiteturas recorrentes, semelhantes as apresentadas na Fig. 2, possuem uma boa capacidade de extrapolação, para o aprendizado de autômatos e linguagens de Tomita 1 e 2.

REFERÊNCIAS

[1] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, The MIT Press, Cambridge MA, ISBN 0-262-63022-2, 1972.

[2] Y. Lecun, Y. Bengio and G. Hinton, *Deep learning. Nature*, vol 521, pp. 436–44, 2015.

[3] I. Sutskever, O. Vinyals, Q. V. V. Le, *Sequence to sequence learning with neural networks.* Advances in Neural Inf .Process. Syst. 27:3104–12, 2014.

[4] H. Sak, A. Senior, F. Beaufays, *Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition*, 2014.

[5] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, *Intriguing properties of neural networks*. Int. Conf. Learn. Represent Apr. 14–16, Banff, Can, 2014.

[6] A. Skrizhevsky, I. Sutskever, G. Hinton, *ImageNet classification with deep convolutional neural networks*. Adv. Neural Inf. Process. Syst. Vol. 25, pp. 1097–105, 2012.

[7] A. Schäfer, H. Zimmermann, *Recurrent Neural Networks are universal approximator*s. Int J Neural Syst., Vol 17, issue 4, pp.253-263, 2007.

[8] A. C. Gil, *Como Elaborar Projetos de Pesquisa*, São Paulo: Editora Atlas S. A., 2002.

[9] R. K. Yin, Estudo de Caso: Planejamento e Métodos, São Paulo: Bookman, 2005.

[10] J. Schmidhuber, *Deep learning in neural networks: An overview*, Neural Networks, 61, 85–117, 2015.

[11] W. Zaremba, T. Mikolov, A. Joulin, R. Fergus, *Learning Simple Algorithms from Examples*, Proceedings of the 33 rd International Conference on Machine Learning, New York, NY, USA, JMLR: W&CP vol. 48, 2016.

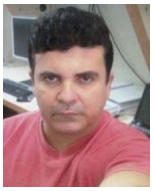
[12] C. Nguyen and M. Nakagawa, *Finite State Machine Based Decoding of Handwritten Text Using Recurrent Neural Networks*, Proceedings of 15th International Conference on Frontiers in Handwriting Recognition, 2016.

[13] A. G. Ororbia II, T. Mikolov, D. Reitter, *Learning Simpler Language Models with the Delta Recurrent Neural Network Framework*, Cornel University Library, arXiv:1703.08864 [cs.CL], 2017.

[14] P. B. Menezes, *Linguagens Formais e Autômatos,* 6ª Ed., Bookman, 2011.

[15] E. O. Nascimento and L. N. Oliveira, Sensitivity Analysis of Cutting Force on Milling Process using Factorial Experimental Planning and Artificial Neural Networks, IEEE Latin America Transactions, vol. 14, no. 12, pp. 4811-4820, 2016.

Lídio Mauro Lima de Campos é Professor Adjunto da Universidade Federal do Pará - UFPA na Faculdade de Computação/ICEN. Doutor em Engenharia Elétrica com ênfase em Computação Aplicada pelo Programa de Pós-Graduação em Engenharia Elétrica - PPGGE na mesma instituição (2016), Mestre em Ciência da Computação, Área de Concentração: Sistemas do Conhecimento, pela Universidade Federal de Santa Catarina - UFSC (2001), Graduado em Engenharia Elétrica pela Universidade Federal do Pará - UFPA (1998) e em Tecnólogo em Processamento de Dados pela Universidade da Amazônia - UNAMA. É Certificado *PMP-Project Management Professional* pelo *Project Management Institute* (PMI-Pennsylvania-EUA). Na área de pesquisa e desenvolvimento atua em Computação Natural, Aprendizagem de Máquina, Inteligência Computacional, Mineração de Dados e Gerência de Projetos.



Alberto Sampaio Lima Doutor em Engenharia de Teleinformática pela Universidade Federal do Ceará. Mestre em Informática Aplicada pela Universidade de Fortaleza. Bacharel em Ciência da Computação pela Universidade Estadual do Ceará. Professor da Universidade Federal do Ceará, com experiência na área de Ciência da Computação, com ênfase em Arquitetura de Sistemas de Computação, atuando principalmente nos seguintes temas: Gestão de Tecnologia da Informação e Comunicação, Redes de Computadores, Engenharia de Software, Inteligência Computacional, Novas Tecnologias na Educação, Avaliação da Educação. Atualmente desenvolve atividades de PÓS-DOUTORADO no Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina -PB.