Pós-Graduação em Ciência da Computação

“Otimização de *Reservoir Computing* com PSO”

Por

Anderson Tenório Sergio

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, MARÇO/2013

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ANDERSON TENÓRIO SERGIO

"Otimização de *Reservoir Computing* com PSO"

Dissertação apresentada à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

ORIENTADORA: Profa. Dra. Teresa Bernarda Ludermir
CO-ORIENTADORA: Profa. Dra. Aida Araújo Ferreira

RECIFE, MARÇO/2013

Catalogação na fonte
Bibliotecária Jane Souto Maior, CRB4-571

Sergio, Anderson Tenório
Otimização de reservoir computing com PSO / Anderson Tenório Sergio. - Recife: O Autor, 2013.
xi, 75 p.: fig., tab.

Orientador: Teresa Bernarda Ludermir.
Dissertação (mestrado) - Universidade Federal de Pernambuco. CIn, Ciência da Computação, 2013.

Inclui bibliografia.

1. Inteligência Computacional. 2. Redes Neurais Artificiais. I. Ludermir, Teresa Bernarda (orientadora). II. Título.

006.3 CDD (23. ed.) MEI2013 – 075

Dissertação de Mestrado apresentada por **Anderson Tenório Sergio** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**Otimização de Reservoir Computing com PSO**" orientada pela **Profa. Teresa Bernarda Ludermir** e aprovada pela Banca Examinadora formada pelos professores:

_____

Prof. Cleber Zanchettin
Centro de Informática / UFPE

_____

Prof. Otoni Nóbrega Neto
Departamento de Engenharia Elétrica / UFPE

_____

Profa. Aida Araújo Ferreira
Instituto Federal de Pernambuco
(Co-orientadora)

Visto e permitida a impressão.
Recife, 7 de março de 2013.

_____

Profa. Edna Natividade da Silva Barros
Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Resumo

Reservoir Computing (RC) é um paradigma de Redes Neurais Artificiais com aplicações importantes no mundo real. RC utiliza arquitetura similar às Redes Neurais Recorrentes para processamento temporal, com a vantagem de não necessitar treinar os pesos da camada intermediária. De uma forma geral, o conceito de RC é baseado na construção de uma rede recorrente de maneira randômica (*reservoir*), sem alteração dos pesos. Após essa fase, uma função de regressão linear é utilizada para treinar a saída do sistema. A transformação dinâmica não-linear oferecida pelo *reservoir* é suficiente para que a camada de saída consiga extrair os sinais de saída utilizando um mapeamento linear simples, fazendo com que o treinamento seja consideravelmente mais rápido. Entretanto, assim como as redes neurais convencionais, *Reservoir Computing* possui alguns problemas. Sua utilização pode ser computacionalmente onerosa, diversos parâmetros influenciam sua eficiência e é improvável que a geração aleatória dos pesos e o treinamento da camada de saída com uma função de regressão linear simples seja a solução ideal para generalizar os dados.

O PSO é um algoritmo de otimização que possui algumas vantagens sobre outras técnicas de busca global. Ele possui implementação simples e, em alguns casos, convergência mais rápida e custo computacional menor. Esta dissertação teve o objetivo de investigar a utilização do PSO (e duas de suas extensões – EPUS-PSO e APSO) na tarefa de otimizar os parâmetros globais, arquitetura e pesos do *reservoir* de um RC, aplicada ao problema de previsão de séries temporais.

Os resultados alcançados mostraram que a otimização de *Reservoir Computing* com PSO, bem como com as suas extensões selecionadas, apresentaram desempenho satisfatório para todas as bases de dados estudadas – séries temporais de *benchmark* e bases de dados com aplicação em energia eólica. A otimização superou o desempenho de diversos trabalhos na literatura, apresentando-se como uma solução importante para o problema de previsão de séries temporais.

Palavras-chave: *Reservoir Computing*, PSO, Otimização, Previsão de Séries Temporais.

Abstract

Reservoir Computing (RC) is a paradigm of Artificial Neural Networks with important applications in the real world. RC uses similar architecture to Recurrent Neural Networks for temporal processing, with the advantage of not needing to train the weights of the hidden layer. In general, the concept of RC is based on constructing a random recurrent network (reservoir) without changing weights. After this phase, a linear regression function is used to train the system output. The non-linear dynamic processing provided by the reservoir is sufficient to extract the output signals from the output layer using a simple linear mapping, so that the training is considerably faster. However, like in conventional neural networks, Reservoir Computing has some problems. The use of RC can be computationally expensive, many parameters influence its effectiveness and it is unlikely that the random generation of the weights and the training of the output layer with a simple linear regression function is the ideal solution to generalize the data.

PSO is an optimization algorithm that has some advantages over other global search techniques. It has simple implementation and in some cases, fast convergence and low computational cost. This work aimed to investigate the use of PSO (and two of its extensions – EPUS-PSO and APSO) in the task of optimizing the global parameters, architectures and weights of a reservoir in RC, applied to the problem of time series forecasting.

The results showed that optimization of Reservoir Computing with PSO, as well as with their selected extensions, had satisfactory performance for all studied databases – benchmark time series and databases with applications in wind energy. The optimization outperformed several works in literature, presenting itself as an important solution to the problem of time series forecasting.

Keywords: Reservoir Computing, PSO, Optimization, Time Series Forecasting.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

AG	Algoritmos Genéticos
AP	*Approximation Property*
APSO	*Adaptive Particle Swarm Optimization*
EPUS-PSO	*Efficient Population Manager for Particle Swarm Optimization*
ESN	*Echo State Networks*
ESP	*Echo State Property*
GSO	*Group Search Optimizer*
LSM	*Liquid State Machines*
MAPE	*Mean Absolute Percentage Error*
MCP	McCulloch-Pitts
MLP	*Multi-Layer Perceptron*
MSE	*Mean Square Error*
NMSE	*Normalized Mean Square Error*
NRMSE	*Normalized Root Mean Square Error*
RC	*Reservoir Computing*
PSO	*Particle Swarm Optimization*
RCT	*Reservoir Computing Toolbox*
RNA	Redes Neurais Artificiais
RNR	Redes Neurais Recorrentes
SA	*Simulated Annealing*
SHI	Sistema Híbrido Inteligente
SP	*Separation Property*
TS	*Tabu Search*

Agradecimentos

Aos meus pais, Edson e Edite, por terem me educado e me oferecido todas as ferramentas para que eu pudesse trilhar o meu caminho, da infância aos tempos mais recentes.

Às minhas orientadoras, Teresa e Aida, pelo apoio, aconselhamento e inspiração profissional.

Aos meus amigos e familiares, pelo incentivo e companheirismo.

Em especial, à minha esposa Aunia, pelo amor, afeto, compreensão e porto seguro. Aproveito para reiterar o que já escrevi em outra oportunidade: obrigado por ser a inspiração de tudo o que faço!

Capítulo 1

Introdução

Este capítulo apresentará a motivação do trabalho, bem como os seus objetivos. Em seguida, a organização da dissertação é exposta.

1.1. Motivação

As Redes Neurais Artificiais (RNA) são uma técnica de inteligência computacional já bastante estabelecida nessa área do conhecimento. Tendo em vista sua capacidade de aprender com exemplos e posteriormente generalizar a informação absorvida, RNA se apresentam como uma solução elegante para resolução de problemas como classificação e reconhecimento de padrões, previsão de séries temporais, modelagem biológica, dentre outros. A razão para seu uso intenso pode ser explicada por algumas das vantagens das RNA frente a outras técnicas que tenham o mesmo objetivo: a não necessidade do conhecimento prévio de um especialista, tolerância a falhas (redes que tenham perdido nodos de sua estrutura podem continuar operantes) e paralelismo.

Entretanto, as Redes Neurais Artificiais apresentam certas limitações. Por exemplo, elas são consideradas “caixas pretas”: é difícil saber como a rede chegou à determinada conclusão, bem como quais itens de sua estrutura são os principais responsáveis por determinada generalização. O treinamento da rede pode ser custoso e a utilização de RNA necessita da definição de parâmetros que variam de acordo com os dados e a aplicação. O projetista precisa definir certas variáveis, como o volume de dados de entrada, número de nodos, número de camadas, melhor estratégia de treinamento, arquitetura, etc. Tais parâmetros só podem ser

determinados pela experiência, através de bom-senso, por tentativa e erro ou mesmo por aprendizado.

Tendo em vista melhorar os resultados de uma Rede Neural Artificial ou mesmo aumentar a eficiência de seu processo de treinamento e utilização, a literatura aponta diversas técnicas de otimização aplicadas nesse tipo de computação inteligente [HSG89] [HBH02] [CFMP99]. Em se tratando de RNA, basicamente existem dois caminhos no processo de otimização. Primeiramente, os chamados métodos de busca global podem tentar automatizar a definição da arquitetura e de outros parâmetros iniciais da rede. Por outro lado, a otimização pode focar na tentativa de encontrar valores adequados para os pesos das conexões, evitando mínimos locais. Diversos algoritmos de busca têm sido utilizados nesse processo de otimização (seja dos pesos da rede ou de sua arquitetura), tais como: Algoritmos Genéticos (AG) [Hol75], *Simulated Annealing*, Busca *Tabu* (TS, do inglês *Tabu Search*) [GL97], GSO (do inglês *Group Search Optimizer*) [HWS09] e Otimização por Enxame de Partículas (PSO, do inglês *Particle Swarm Optimization*) [KE95].

Reservoir Computing é um paradigma de Redes Neurais Artificiais com aplicações importantes [VFV+10] [BVV+11] [SDP+12]. RC utiliza arquitetura similar a Redes Neurais Recorrentes (RNR) para processamento temporal sem a dificuldade de treinar a camada intermediária da rede. *Reservoir Computing* foi introduzido independentemente como *Liquid State Machine*s (LSM) [MNM02] e *Echo State Networks* (ESN) [Jae01]. De forma geral, o conceito de RC é baseado na construção de uma RNR de maneira randômica (essa camada é denominada *reservoir*), sem alteração dos pesos. Após essa fase, uma função de regressão linear é utilizada para treinar a saída do sistema. Schrauwen et al [SDV+07] mostram que a transformação dinâmica não-linear oferecida pelo *reservoir* é suficiente para que a camada de saída, denominada *readout*, consiga extrair os sinais de saída utilizando um mapeamento linear simples.

Assim como ocorre nas redes neurais convencionais, *Reservoir Computing* sofre de algumas desvantagens. Por se tratar de uma abordagem de redes recorrentes, sua utilização pode ser computacionalmente onerosa, mesmo sem a fase de treinamento da camada de *reservoir*. Diversos parâmetros influenciam a eficiência do RC, como, por exemplo, o número de nodos utilizados e o tipo de

função de ativação. Definir esses parâmetros sem a experiência na resolução de determinado problema é difícil. Lukosevicius e Jaeger [LJ09] indicam que é improvável que a geração aleatória dos pesos e o treinamento da camada de saída com uma função de regressão linear simples seja a solução ideal para computar RC.

Assim, a utilização de um algoritmo de otimização para melhorar o desempenho de *Reservoir Computing* é um campo ainda em aberto na literatura. Diversas soluções têm sido propostas, seja como alternativa para geração do *reservoir* ou treinamento do *readout*. Ferreira desenvolveu um método para encontrar o melhor *reservoir* aplicado à tarefa de prever séries temporais, denominado RCDESIGN, utilizando Algoritmos Genéticos [Fer11]. O RCDESIGN busca uma melhor configuração de parâmetros, arquitetura e pesos do *reservoir* de acordo com alguns casos particulares. Os resultados mostraram que o RCDESIGN melhorou o desempenho do RC, além de apresentar-se como alternativa de menor custo computacional quando comparado a outros trabalhos.

1.2. Objetivos

O PSO [KE95] é um algoritmo de otimização que possui algumas vantagens sobre outras técnicas de busca global. O algoritmo é baseado no comportamento social de revoadas dos pássaros: uma população de soluções é mantida e cada indivíduo busca melhorar seu desempenho baseado na melhor experiência pessoal e na melhor experiência do grupo. Quando comparado aos Algoritmos Genéticos, o PSO possui a vantagem de ter implementação mais simples e, em alguns casos, convergência relativamente mais rápida e custo computacional menor [HCdWV05] [PP07].

Esta dissertação tem o objetivo de investigar a utilização do PSO na tarefa de otimizar os parâmetros, arquitetura e pesos do *reservoir* de um RC, aplicada ao problema de previsão de séries temporais. Utilizar o PSO nessa tarefa, incluindo a otimização dos pesos, é inédito na literatura. Além do PSO padrão, também será estudada a utilização de duas de suas extensões (EPUS-PSO e APSO, ambos algoritmos discutidos no capítulo 3).

Para validação do método proposto, serão utilizadas sete séries temporais clássicas e três séries de velocidade média horária dos ventos na região Nordeste

do Brasil. As séries de *benchmark* serão utilizadas para comparação com outros trabalhos na literatura, enquanto que as séries de ventos verificarão a adequação do método em um problema de aplicação prática, com relevância econômica.

1.3. Organização da Dissertação

Neste primeiro capítulo, foram apresentadas a motivação e os objetivos da dissertação. A seguir, a organização dos demais capítulos:

- Capítulo 2 – *Reservoir Computing*

  Este capítulo aborda em um primeiro momento as Redes Neurais Artificiais, de acordo com suas principais características. Após uma breve apresentação das Redes Neurais Recorrentes, o *Reservoir Computing* é apresentado, segundo seus dois principais métodos e aplicações.
- Capítulo 3 – *Particle Swarm Optimization* - PSO

  O capítulo 3 expõe o método de otimização utilizado neste trabalho, o PSO. Após a apresentação do algoritmo do PSO, são descritas duas das suas extensões presentes na literatura, abordando suas diferenças e potencialidades comparadas com o algoritmo original.
- Capítulo 4 – Otimização de *Reservoir Computing*

  O capítulo 4 revisa, através de aplicações na literatura, como o desempenho das Redes Neurais Artificiais pode ser melhorado com a utilização de algoritmos de otimização. Em seguida, este mesmo conceito é aplicado a *Reservoir Computing*, sendo discutido de que forma o desempenho de RC pode ser melhorado.
- Capítulo 5 – Otimização de *Reservoir Computing* com PSO

  Neste capítulo, o método proposto para realizar otimização de *Reservoir Computing* com PSO é apresentado. Em seguida, o método experimental é descrito. As bases de dados utilizadas nas simulações numéricas e as abordagens para comparação com outros trabalhos na literatura são discutidas.
- Capítulo 6 – Simulações Numéricas

  O capítulo de Simulações Numéricas expõe os resultados alcançados pelo método proposto nas bases de dados previamente apresentadas, com sua

devida análise. Então, os resultados são comparados com diversas abordagens presentes na literatura.

- Capítulo 7 – Conclusões e Trabalhos Futuros
  O capítulo 7 apresenta as conclusões alcançadas a partir dessa dissertação. Em seguida, os possíveis trabalhos futuros são listados.

Capítulo 2

Reservoir Computing

Neste capítulo, as Redes Neurais Artificiais são apresentadas, bem como os conceitos envolvidos para que os pesquisadores chegassem à definição de *Reservoir Computing*. RC é descrito de acordo com suas principais arquiteturas, processo de treinamento e aplicações.

2.1. Redes Neurais Artificiais

Redes Neurais Artificiais são sistemas paralelos distribuídos compostos por unidades de processamento simples que calculam funções matemáticas. As unidades (chamadas nodos ou neurônios) são dispostas em camadas e interligadas por conexões. As conexões são associadas a determinados pesos. RNA podem ser usadas em tarefas como classificação e reconhecimento de padrões, previsão de séries temporais e modelagem biológica.

RNA têm inspiração biológica e são baseadas na forma como o cérebro processa informações e gera saídas a partir de certos estímulos. O modelo hoje conhecido como McCulloch-Pitts (MCP), definido por Warren McCulloch e Walter Pitts [MP43], é uma simplificação do funcionamento de um neurônio biológico. A Figura 1 apresenta o esquema de um neurônio biológico seguida de sua representação matemática proposta por estes pesquisadores.

O neurônio biológico, de forma simplificada, recebe estímulos de outras unidades conectadas a ele através dos dendritos. O sinal é então processado no corpo celular e transmitido através do axônio. No modelo artificial, existem n terminais de entrada x_1, x_2, ..., x_n, representando os dendritos. Para representar o axônio, tem-se o terminal de saída y. Quanto as sinapses (conexões entre diferentes neurônios), para cada entrada existe um peso associado w_1, w_2, ..., w_n. Assim, a

emulação do corpo do neurônio recebe a soma das entradas ponderadas por seus respectivos pesos. Para simular o chamado limiar de excitação do neurônio biológico, a saída do modelo artificial só é ativada quando um determinado *threshold* é atingido. A representação matemática do modelo MCP é dada pela Equação 2.1:

$$\sum_{i=1}^{n} x_i w_i \geq \theta \tag{2.1}$$

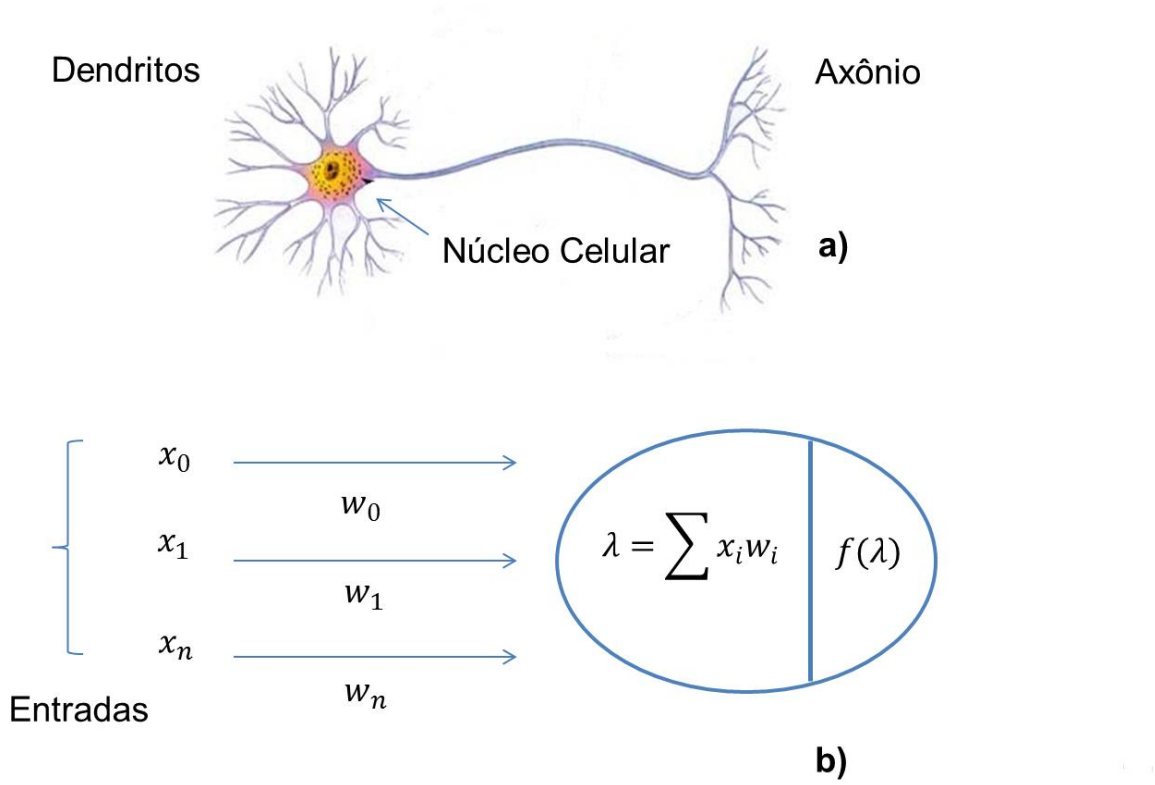

Figura 1. Representações de neurônios: a) neurônio biológico b) neurônio artificial

O modelo MCP original tem saída binária: o neurônio artificial está ativo ou não. Ao longo dos anos, outros modelos foram propostos para que a unidade de processamento produzisse uma saída qualquer, através da adoção de funções de ativação não lineares [Hay99]. Algumas das funções de ativação utilizadas podem ser vistas na Figura 2.

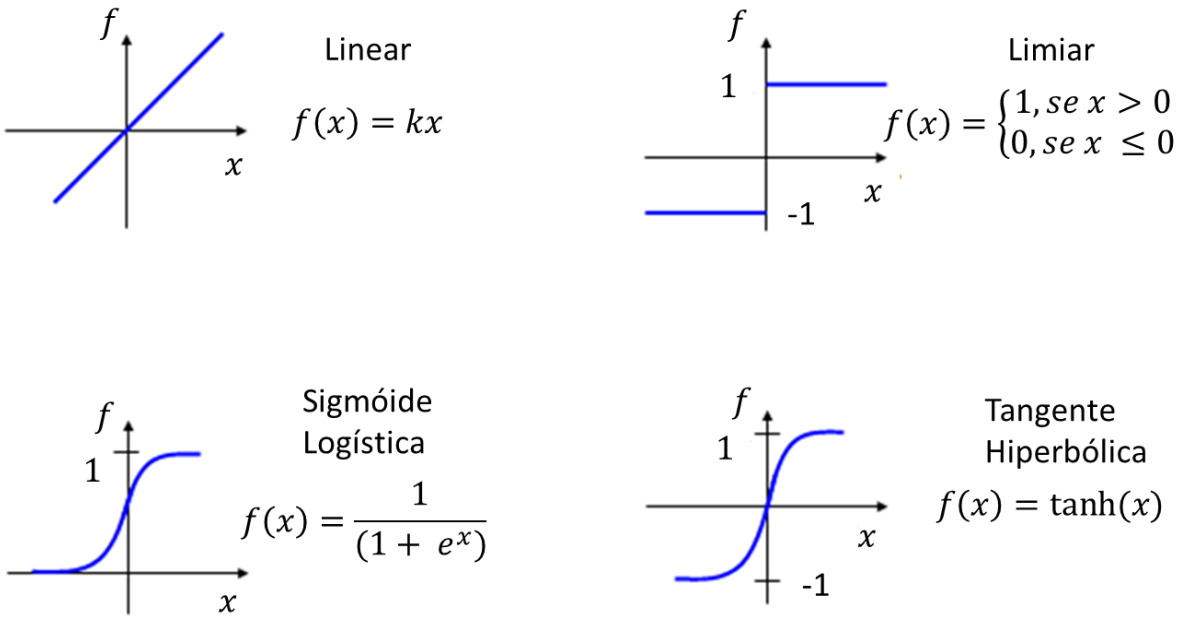

Figura 2. Exemplos de funções de ativação: linear, limiar, sigmoide logística e tangente hiperbólica

Comumente, a principal vantagem das Redes Neurais Artificiais é sua capacidade de aprender através de exemplos e generalizar as informações aprendidas, em contrapartida a um simples mapeamento entrada-saída. O processo inicial é denominado treinamento. Generalizar diz respeito à capacidade da rede de apresentar respostas coerentes a dados nunca antes utilizados no treinamento, fazendo interpolações e extrapolações do que aprenderam.

A etapa de treinamento (ou aprendizado) é constituída pela execução de um algoritmo de aprendizado. O algoritmo de aprendizado é um conjunto de procedimentos bem-definidos para adaptar os parâmetros de uma RNA para que ela possa aprender uma determinada função [BCL07]. Na fase de aprendizagem, a rede cria uma representação própria para o problema, através da extração de informações relevantes dos padrões de entrada apresentados a ela.

No processo de aprendizagem de uma RNA, diversos métodos podem ser utilizados. Os dois principais paradigmas que agrupam tais métodos são o aprendizado supervisionado e o aprendizado não supervisionado. No primeiro, a entrada e a saída desejadas são fornecidas por um supervisor (ou professor) externo, com o objetivo de ajustar os parâmetros da rede para encontrar uma

relação entre os pares de entrada e saída. Já no caso do aprendizado não supervisionado, não há a figura de um professor. Nessa abordagem, a rede desenvolve a habilidade de formar representações internas para codificar características da entrada. Outros paradigmas conhecidos são o aprendizado por reforço, que é um caso particular do aprendizado supervisionado e o aprendizado por competição, uma derivação do aprendizado não supervisionado.

Haykin [Hay99] lista as principais propriedades úteis e capacidades das RNA.

- Não linearidade: as redes podem ser compostas por neurônios lineares ou não lineares.
- Mapeamento de entrada-saída: as redes possuem a capacidade de, a partir de um conjunto de treinamento, estabelecer uma relação de paridade entre estímulos e respostas.
- Adaptabilidade: usualmente, uma rede neural treinada para resolver um determinado problema tem a capacidade de se adaptar a outro, a partir de um "retreinamento".
- Resposta a evidências: em classificação de padrões, uma rede neural pode oferecer, além de qual padrão selecionar, o grau de confiança ou crença naquela resposta.
- Informação contextual: nas redes neurais, a informação contextual é tratada naturalmente, já que o conhecimento é representado por sua própria estrutura e cada neurônio é afetado pelos demais nodos.
- Tolerância a falhas: devido à natureza distribuída da informação armazenada na rede, um dano em um ou mais neurônios em uma implementação física (em hardware) pode representar apenas uma degradação suave na rede como um todo.
- Implementação em VLSI (do inglês, V*ery-Large-Scale-Integration*): devido a sua natureza paralela, as redes neurais são adequadas para implementação utilizando tecnologia VLSI, tornando-se apta a realizar computação de certas tarefas com rapidez.
- Uniformidade de análise e projeto: as redes neurais são universais como processadores de informação. Os neurônios são ingredientes comuns a todas

as abordagens, fazendo com que se possam compartilhar teorias e algoritmos de aprendizagem em aplicações diferentes.

- Analogia neurobiológica: as redes neurais são motivadas pela analogia com o cérebro, que é uma prova viva de que o processamento paralelo e tolerante a falhas é fisicamente possível, rápido e poderoso.

Em contrapartida aos atrativos oferecidos pelas Redes Neurais Artificiais, certas desvantagens podem ser observadas. De um modo geral, os resultados alcançados por uma RNA podem ser considerados uma “caixa preta”: os critérios para a tomada de decisão não são claros e é difícil descobrir quais elementos de sua estrutura são responsáveis por determinado comportamento. Adicionalmente, algumas fases do processo de treinamento e uso de uma rede neural são constituídas por decisões empíricas, onde não há regras para a definição do volume de dados de entrada e melhor estratégia de treinamento. Certos parâmetros são determinados pelo simples bom-senso.

Além disso, certos algoritmos de aprendizagem podem cair nos chamados mínimos locais, como o *backpropagation* [Hay99], utilizado em redes do tipo *feedforward*. Esse algoritmo, por exemplo, utiliza a técnica do gradiente descendente para ajustar os pesos das conexões através de pares entrada-saída sem qualquer conhecimento prévio do problema. O processo pode levar o treinamento para pontos indesejáveis do espaço de busca. Para aumentar a probabilidade de convergência, evitando cair em mínimos locais, algumas estratégias têm de ser tomadas, como pode ser visto em [AS07].

2.2. Redes Neurais Recorrentes

Redes Neurais Recorrentes são uma classe de RNA que possuem pelo menos uma conexão de *feedback*. Essa característica faz com que RNR sejam capazes de criar um estado interno que as permitem apresentar comportamento temporal dinâmico. Segundo esse aspecto, tais redes são diferentes das Redes Neurais Artificiais do tipo *feedforward*. Nas RNA tradicionais as conexões entre suas unidades de processamento não formam ciclos de realimentação (ou *feedback*), sendo este o caso da consagrada arquitetura MLP (do inglês, *Multi-Layer Perceptron*) [Hay99].

Os fenômenos temporais são uma das motivações para a utilização de Redes Neurais Recorrentes. O fator principal de tais fenômenos, o tempo, é crucial para a realização de algumas tarefas importantes da atividade humana, por exemplo: reconhecimento de padrões, reconhecimento de voz, detecção de movimento, processamento de sinais, etc. As redes do tipo *feedforward* e seus diversos algoritmos de treinamento apresentam dificuldades em implementar mapeamentos dinâmicos. Nas redes *feedforward*, uma solução para realizar o processamento temporal (apesar de não ser muito indicada) é a utilização de janelas de tempo, onde a rede utiliza trechos de dados temporais como se eles seguissem um padrão estático.

Para uma RNA ser considerada realmente dinâmica, tendo assim a capacidade de incorporar fenômenos temporais, ela deve possuir memória [Elm90]. Assim, essas redes tornam-se sensíveis a sinais que variam com o tempo. Existem basicamente duas formas de incorporar memória em uma Rede Neural Artificial [BCL07]:

- Introduzir atraso no tempo. Exemplos desse tipo de arquitetura: TDNN (do inglês *Time Delay Neural Network*) [LH88] [Wai89] [WHHSL89] e FIR *multilayer perceptron* [Wan90a] [Wan90b].
- Utilizar redes recorrentes. Exemplos de RNR: *Backpropagation Through Time* [Wer74] [Wer90] [WP89], *Real-Time Recurrent Learning* [WZ89], *Cascate Correlation* Recorrente [Fah91], redes de Elman [Elm90] e redes de Jordan [Jor86].

A Figura 3 mostra um esquema básico de uma Rede Neural Recorrente. Importante observar a presença de uma conexão de *feedback*, diferenciando-a de uma rede do tipo *feedforward*.

Em resumo, devido à capacidade de possuir conexões de realimentação, as RNR possuem maiores possibilidades de combinar as informações provenientes dos sinais de entrada. Sendo assim, em geral precisam de menos unidades de processamento quando comparadas às redes *feedforward*. As RNR são mais semelhantes às redes biológicas, fazendo delas melhores candidatas para o entendimento do cérebro [Hay99].

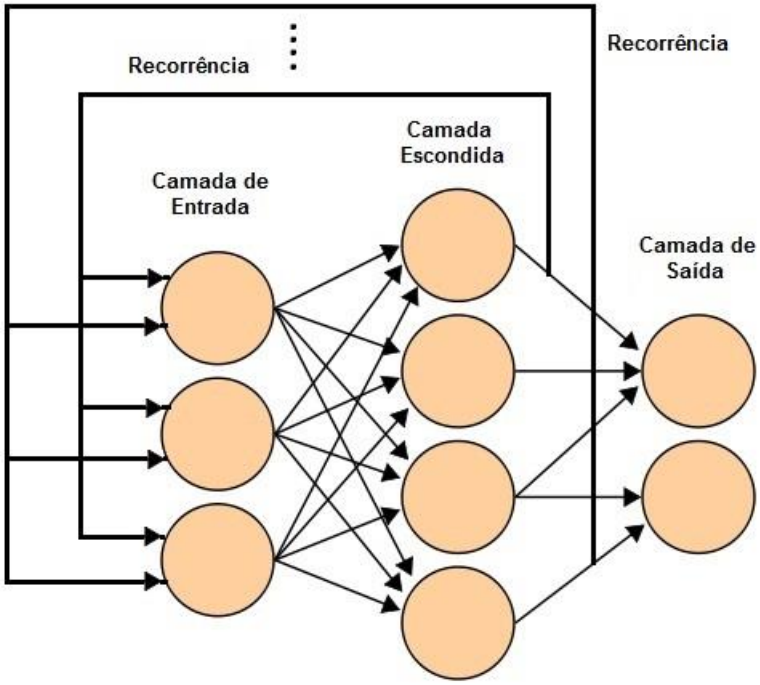

Figura 3. Exemplo de um diagrama de uma Rede Neural Recorrente

Entretanto, alguns problemas com as Redes Neurais Recorrentes podem ser observados. RNR são mais complexas do que redes do tipo *feedforward*, sendo o seu treinamento mais difícil. Além disso, muitas abordagens dessa categoria de redes neurais sofrem problema de escala [BCL07]. Isto é, a dificuldade de treinar a rede aumenta consideravelmente com um grande número de unidades de processamento ou mesmo sinais de entrada. Atualmente, treinar Redes Neurais Recorrentes de maneira eficiente e sem problemas de escala é um campo de pesquisa em aberto [BSF94].

2.3. Reservoir Computing

Reservoir Computing é uma arquitetura de Redes Neurais Artificiais que foi desenvolvida independentemente como *Liquid State Machines* [MNM02] e *Echo State Networks* [Jae01]. Em comum, ambos os modelos têm a característica de utilizar o poder computacional das Redes Neurais Recorrentes sem as dificuldades relacionadas ao treinamento dessa arquitetura.

De um modo geral, a rede é composta por uma rede recorrente com um número relativamente grande de unidades de processamento, denominada

reservoir. O *reservoir* recebe os sinais de entrada e envia-os para um circuito menor, chamado *readout*. Enquanto os pesos do *reservoir* são definidos randomicamente no início do processo e mantidos inalterados, o *readout* é utilizado para treinar a saída da rede através de uma função que faça regressão linear dos sinais provenientes da camada intermediária. A Figura 4 mostra o diagrama de uma *Echo State Network* simples.

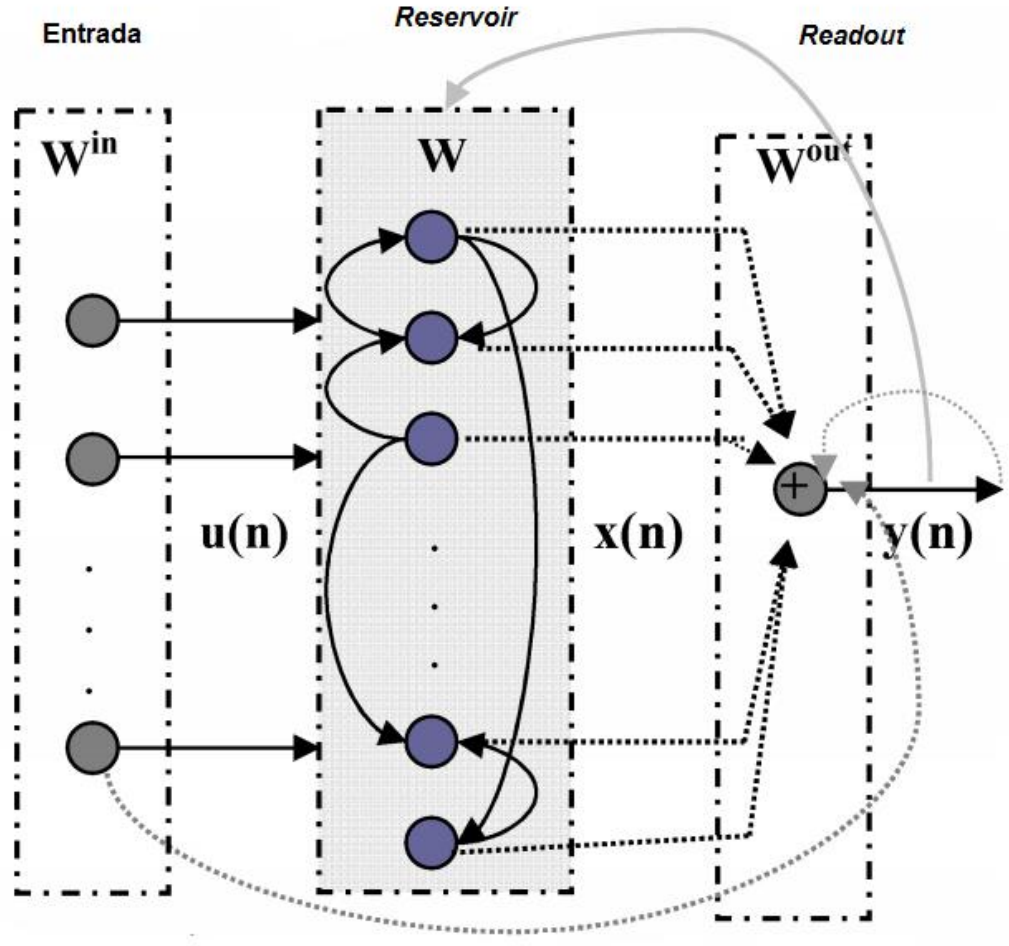

Figura 4. Arquitetura de uma *Echo State Network*

Como explicitado na Figura 4, a camada denominada *reservoir* recebe como sinal os valores advindos da camada de entrada, bem como opcionalmente os sinais provenientes da conexão de *feedback* e de bias. O *reservoir*, com certo número de unidades de processamento, é projetado de forma que possua conexões recorrentes. Os pesos dessas conexões são randômicos e não mudam. A camada denominada *readout* faz um mapeamento linear simples da saída do *reservoir*

utilizando, por exemplo, pseudo-inversa [LJ09]. Ainda na Figura 4, as linhas pontilhadas representam conexões que podem ser treinadas e as linhas sombreadas indicam conexões opcionais.

A ideia principal de um *Reservoir Computing* é pôr a rede em um estado onde os neurônios ativam uns aos outros, conduzindo-a a um estado estável imutável. O sinal de entrada percorre este caminho de ativação até se dissiparem, quando os pesos das conexões vão progressivamente diminuindo. Como esses rastros são lentos o suficiente e podem ser distinguíveis entre si, o sistema cria a capacidade de possuir memória [Jae01]. Os sinais “ecoam” até se extinguirem e esses ecos são usados para a computação (origem do termo *Echo State Network*). Por outro lado, o processo descrito pode ser lembrado como as ondas na superfície de um lago que vão desvanecendo após este ser estimulado por um objeto lançado sobre ele (origem do termo *Liquid State Machine*) [MNM02]. Construir uma rede que possui ecos úteis é significantemente mais fácil do que criar uma que possua um conjunto particular de estados estáveis [Jae01].

Outro ponto relacionado à motivação de utilizar *Reservoir Computing* vem da tentativa de encontrar uma alternativa para o treinamento de Redes Neurais Recorrentes, consideravelmente mais complexo do que em redes do tipo *feedforward*. Ao invés de tentar obter uma transformação particular através do ajuste dos pesos, RC utiliza um número maior de neurônios (em comparação às redes convencionais) para alcançar um conjunto diverso de transformações a partir dos sinais de entrada [Jae01]. Geralmente tais transformações não são as desejadas, mas elas podem ser combinadas para alcançá-las. Esta ação é justamente realizada pela camada de *readout*. O cálculo realizado é simples, já que o *readout* não apresenta retroalimentação. É importante notar que um mesmo *reservoir* pode ser utilizado para calcular múltiplas transformações em paralelo, desde que elas tenham diferentes *readouts*.

2.4. Métodos de *Reservoir Computing*

A seguir, serão apresentados os conceitos-chave dos dois trabalhos que originaram a arquitetura hoje conhecida como *Reservoir Computing*, *Liquid State Machines* e *Echo State Network*. Busca-se compreender a dinâmica envolvida e perceber como esse tipo de Rede Neural Artificial pode ser utilizada.

2.4.1. *Liquid State Machines*

Como explicitado anteriormente, RNA têm inspiração biológica. O cérebro dos animais possui uma intricada rede de vias *feedforward* que processam as informações advindas do ambiente no chamado neocórtex. Apesar de essas redes biológicas apresentarem um poder de processamento aparentemente universal, suas unidades de processamento são simples. O nível de complexidade da rede é principalmente constituído por múltiplos *loops* recorrentes em mais de 80% das sinapses dentro de uma coluna neocortical. É justamente nesse comportamento que reside a dificuldade em modelar computacionalmente essa característica natural [Sav98], baseada em redes recorrentes e neurônios do tipo *Spiking* [Bur05b].

Outro problema na modelagem computacional de redes neurais biológicas é o alto grau de dimensionalidade a que elas estão submetidas. Os componentes são heterogêneos e cada um dos neurônios e sinapses adicionam um grau de liberdade no sistema. É justamente na tentativa de controlar esse alto grau de dimensionalidade que a maioria das abordagens da modelagem computacional de circuitos neurais estão inseridas. Pearlmutter [Pea95] revisa alguns modelos artificiais que constroem mecanismos adaptativos para controlar essa dinâmica. Outros modelos, baseados na máquina de Turing (estes podem ser vistos em [Pol91], [GMC+92], [SS94], [Hyo96] e [Moo98]), até são capazes de realizar computação em tempo real e funcionar como reconhecedores dinâmicos. Contudo, esbarram em uma importante característica: abordagens baseadas na máquina de Turing requerem um relógio para sincronizar as operações dos neurônios, e esse comportamento parece não estar presente nos modelos biológicos. Adicionalmente, tais modelos não são capazes de se adaptar a circuitos pré-existentes, sendo necessário criar sua própria rede recorrente.

Há ainda outra abordagem na modelagem computacional de redes neurais biológicas, as chamadas redes neurais atratoras [Ami89]. Tais redes têm a vantagem de permitir computação robusta baseada em ruídos (características presente nos modelos naturais), mas requerem um alto número de unidades atratoras para armazenar informações do passado.

Nenhuma dessas abordagens tem a capacidade de permitir a computação em tempo real ao mesmo tempo em que tornem suas operações paralelas. Foi com o objetivo de construir um modelo com essas características que Maass, Natschläger e Markran [MNM02] propuseram as *Liquid States Machines*. LSM focam no comportamento de um neurônio do tipo *readout*. A tarefa de um neurônio *readout* é extrair informações de um circuito e transmiti-las para outros. Essa unidade de processamento recebe entradas de milhares de outros neurônios, extraem informações do alto grau de dimensionalidade de um estado transitório e convertem essas informações em estados estáveis. Múltiplos neurônios *readouts* podem ser treinados para realizarem diferentes tarefas, gerando assim um sistema paralelo. Algumas das ideias presentes nas *Liquid States Machines* já estavam presentes nos trabalhos de Buonomano [BM95], Dominey [DAJ95] e Jaeger [Jae01].

Para entender o conceito de *Liquid States Machines*, pode-se realizar uma analogia com uma entidade do mundo real, o líquido. O líquido pode ser considerado como um sistema composto por uma série de perturbações externas passageiras em um meio excitável. Essas perturbações podem ser o vento, o som ou uma sequência de pedras atiradas no líquido, por exemplo. Fazendo uma comparação com as redes neurais atratoras, o líquido possui apenas um estado estável (quando em repouso), o que não representa qualquer informação sobre as entradas passadas. Entretanto, em qualquer momento no tempo, o estado perturbado no líquido representa tanto o presente como o passado das entradas a que foi submetido, podendo prover uma análise da dinâmica do ambiente, uma “análise retrógrada”. Nesse sentido, Gupta [GWM00] mostra que microcircuitos neurais parecem ser líquidos ideais para computação sobre estados perturbados devido a grande variedade de unidades de processamento e conexões.

Liquid States Machines foram propostas como um modelo computacional sem estados estáveis, baseado principalmente em duas propriedades macroscópicas: *Separation Property* (SP) e *Approximation Property* (AP) [MNM02]. A propriedade SP está relacionada com a separação entre as trajetórias dos estados internos do sistema por duas diferentes entradas. Por outro lado, a propriedade AP relaciona-se com a capacidade de codificação dos mecanismos do tipo *readout*, ou seja, sua capacidade de transformar estados internos em saídas do sistema.

Tendo todas essas características em vista, a Figura 5 mostra a arquitetura de uma LSM.

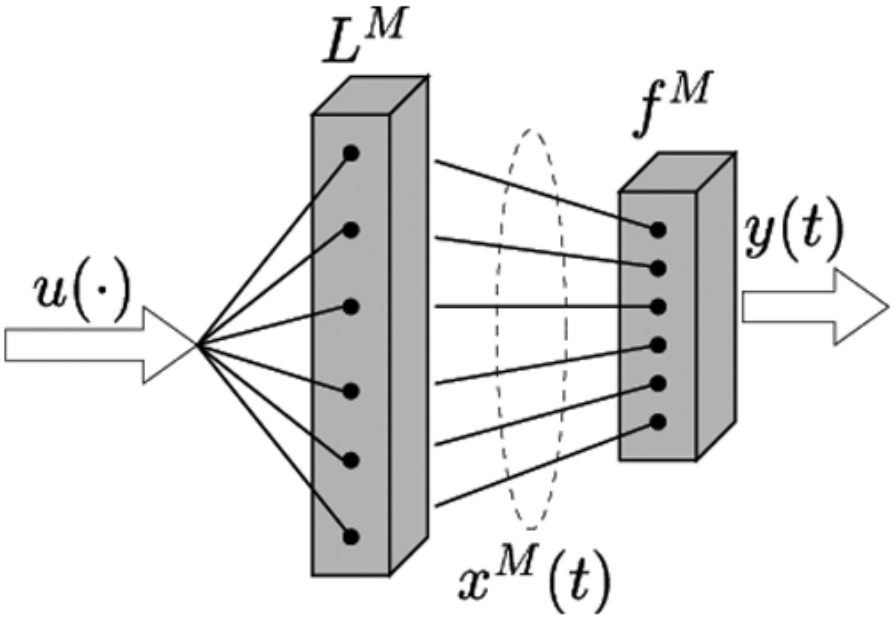

Figura 5. Arquitetura de uma *Liquid State Machine*

Uma sequencia contínua de perturbações caracteriza a função de entrada *u*(.), enquanto que *y(t)* representa alguma função de tempo escolhida como saída que provê uma análise em tempo real da sequência. A cada instante de tempo *t*, uma máquina *M* com função de mapear *u*(.) em *y(t)* gera um "estado líquido" interno, denominado $x^M(t)$. Esse estado líquido consiste de valores que podem mudar continuamente ao longo do tempo, diferentemente de uma máquina de estado finito. Adicionalmente, os estados líquidos e as transições entre eles não precisam ser customizados para uma determinada tarefa. O estado líquido é gerado pelo filtro líquido (ou circuito líquido), L^M. Esse filtro pode ser implementado por um circuito neural, denominado nesse caso de neurônios líquidos. O segundo componente de um LSM é a função sem memória *readout* (apresentada na Figura 5 como f^M), que transforma a cada instante de tempo *t* o estado líquido atual na saída desejada. Em contraste com o filtro líquido, a função *readout* é construída para realizar determinada tarefa, podendo haver inclusive várias dessas funções. O fato de poderem existir diferentes funções *readouts* para tarefas distintas é o que gera o poder de paralelismo de um LSM. O artigo que apresenta as *Liquid States Machines*

mostra todo o embasamento matemático para essa arquitetura, provando ser um aproximador universal de funções [MNM02].

Liquid States Machines podem ser consideradas um aperfeiçoamento no campo das Redes Neurais Artificiais pelos seguintes motivos:

- Os circuitos não são construídos para a realização de uma tarefa em específico, diferentemente dos modelos inspirados na máquina de Turing.
- Entradas contínuas no tempo são tratadas de maneira mais natural, buscando assemelhar-se com o que acontece nas redes biológicas.
- A mesma rede pode ser utilizada para realizar computações múltiplas. Essa característica inclusive provê ideias para a implementação dessa arquitetura em circuitos do tipo VLSI (*Very-Large-Scale-Integration*), aproveitando-se de sua capacidade de paralelismo.

Por outro lado, algumas de suas desvantagens podem ser levantadas:

- Há pouco controle sobre os processos envolvidos.
- LSM não necessariamente emula nem explica as funções cerebrais. O que essa arquitetura faz é replicar algumas características do modelo biológico.
- É uma “caixa preta”, não há garantias de que se entenda que tipo de computação esteja sendo realizada nos componentes internos da arquitetura.
- São ineficientes do ponto de vista de implementação, quando comparadas com circuitos projetados ou mesmo Redes Neurais Artificiais.

2.4.2. *Echo State Networks*

ESN são uma arquitetura de Redes Neurais Recorrentes com aprendizado supervisionado. A ideia principal dessa abordagem é imprimir um sinal de entrada a uma rede neural recorrente fixa, grande o suficiente para o problema e randômica (camada denominada *reservoir*), e treinar o sinal de resposta dessa camada através de combinação linear. ESN foram propostas por Herbert Jaeger em 2001 [Jae01]. Muitas das ideias dessa arquitetura são compartilhadas com as *Liquid State Machines*, mas ambos os trabalhos foram conduzidos independentemente.

A motivação para o desenvolvimento das ESN originou-se de algumas desvantagens da utilização de Redes Neurais Recorrentes, geralmente treinadas com o gradiente descendente sobre uma função de erro. Outros algoritmos têm sido utilizados na literatura, entretanto todas essas abordagens apresentam certas características desfavoráveis: mínimos locais; convergência lenta; perturbações e desaceleração do processo de aprendizagem quando a rede é conduzida através de bifurcações durante o aprendizado; dificuldades para aprender dependências temporais mais acentuadas devido à fuga ou explosão das estimativas do gradiente e a relativa complexidade de todos esses métodos, prejudicando seu uso em larga-escala [Jae01].

Para evitar essas desvantagens, quando um sistema dinâmico deve ser representado por uma rede neural, alguns caminhos podem ser tomados:

- Treinar uma rede neural *feedforward* para predizer uma outra rede cuja entrada são algumas instâncias atrasadas da entrada e/ou saída do sistema.
- Utilizar arquiteturas restritas de Redes Neurais Recorrentes com algoritmos de aprendizagem conhecidos, como, por exemplo, as bem estabelecidas Redes de Elman [Elm90].
- Utilizar arquiteturas de redes com *design* customizável para sobrepor alguns dos problemas apresentados. Um exemplo disso são as redes "*Long Short Term Memory*" [JSC00].

Todas essas características apontadas acima vão de encontro às soluções apresentadas pelas redes neurais biológicas. Tais redes são intrinsicamente recorrentes, com aprendizado versátil e eficiente. Levando esse cenário em consideração, as *Echo State Networks* foram propostas. De forma geral, a computação em ESN pode ser dividida em alguns passos bem definidos:

- Criar uma Rede Neural Recorrente randômica, utilizando qualquer modelo de neurônio artificial (McCullough-Pitts ou *Spiking* [Bur05b]). Esse componente é denominado *reservoir*.
- Criar conexões *all-to-all* entre a camada de entrada e o *reservoir* utilizando pesos aleatórios.

- Criar neurônios de saída e as conexões do *reservoir* para a camada de saída.
- Treinar os pesos de saída por um método de regressão linear. Esse método não é fixado pelo algoritmo, podendo ser, por exemplo, LMS (*Learning Management System*) [BHOM09] ou SVM (*Support Vector Machines*) [MLH03].

A Figura 6 mostra a arquitetura de uma *Echo State Network*.

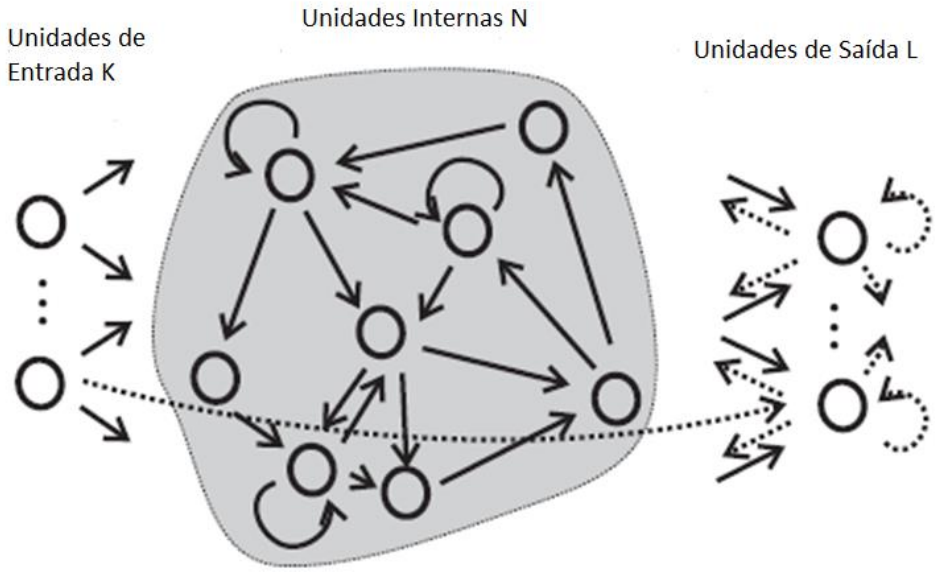

Figura 6. Arquitetura de uma *Echo State Network*

A mudança de estado em uma rede com N unidades no *reservoir*, *K* entradas e *L* unidades de saída é governada pela Equação 2.2:

$$x(n+1) = f\left(Wx(n) + W^{in}u(n+1) + W^{back}y(n)\right) \quad (2.2)$$

A expressão *x(n)* é o estado do *reservoir*. *f* é alguma função sigmoide, podendo ser a logística ou a tangente hiperbólica. W, W^{in} e W^{back} são as matrizes de peso do *reservoir*, entrada e *feedback*, respectivamente. *u(n)* é o sinal de entrada, enquanto que *y(n)* é o sinal de saída. O estado estendido do sistema é dado por *z(n)* (Equação 2.3), sendo a concatenação do estado da rede *x(n)* e da entrada *u(n)*. A saída da rede é calculada conforme Equação 2.4. W^{out} é a matriz de

pesos da camada de saída e *g* é a função de ativação dos neurônios da camada de saída (identidade ou sigmoide).

$$z(n) = [x(n): u(n)] \tag{2.3}$$

$$y(n) = g(W^{out} z(n)) \tag{2.4}$$

É importante ressaltar que, para que a *Echo State Network* comporte-se adequadamente, o *reservoir* deve apresentar o chamado ESP (*Echo State Property*) [Jae01]. O ESP está relacionado com propriedades assintóticas da dinâmica do *reservoir* no que diz respeito à sua excitação pelos sinais de entrada. Quando esta condição é estabelecida, é somente necessário adaptar os pesos da saída para obter alto desempenho. Assim como as *Liquid States Machines*, as *Echo State Networks* também são aproximadores universais de funções [Jae01].

ESN têm sido utilizadas em uma grande variedade de tarefas, tais como: geradores de ondas senoidais sintonizáveis, reconhecimento de formas de onda, reconhecimento de palavras, reconhecimento de padrões dinâmicos, dispositivos de medidores de frequência, atratores caóticos, entre outras [Jae01]. *Echo State Networks* também têm capacidade de ser uma memória de curto prazo. De acordo com o artigo que propôs essa arquitetura, a capacidade de memória de uma ESN é delimitada pelo número de unidades da rede. Alguns pontos de pesquisa relacionados à utilização de *Echo State Networks* podem ser destacados. Um deles diz respeito à estabilidade das ESN: redes dessa arquitetura são comprovadamente estáveis sem *feedback*, porém, com a adição de *feedback*, essa estabilidade pode ser perdida [Jae01]. Outro ponto diz respeito à utilização de ESN hierárquicas ou modulares [Jae07].

2.5. Aplicações

Aplicações de sucesso na utilização de *Reservoir Computing* podem ser observadas tanto na resolução de problemas abstratos como aplicações em engenharia no mundo real. A seguir, algumas dessas aplicações:

- Classificação de padrões dinâmicos [Jae02];

- Geração de sinos autônomos [Jae01];
- Computação de funções não-lineares em taxas instantâneas de trens pulsantes [MNM04];
- Utilização de LSM para controlar um braço de um robô simulado [JM04], modelar um controlador de robô [Bur05b] e executar a previsão da trilha e do movimento de um objeto [Bur05a] [MLM02];
- Identificação de eventos [Jae05];
- Aplicações na competição *Robocup* [SP05] [OLSB05];
- Modelagem de ruído [JH04];
- Geração e previsão de séries temporais caóticas [Jae02] [Jae03] [Ste05] [Ste06];
- Processamento de informações ópticas [VFV+10];
- Detecção de imagens em eletroencefalogramas [BVV+11];
- Previsão da velocidade dos ventos e da geração eólica [Fer11];
- Reconhecimento de fala [MNM03] [VSSC05] [SDP+12].

Inclusive, em algumas aplicações como previsão de séries temporais caóticas e reconhecimento de dígitos isolados, técnicas de *Reservoir Computing* são consideradas estado-da-arte [SVH12] [LJ09].

Capítulo 3

Particle Swarm Optimization - PSO

Este capítulo apresenta o algoritmo de otimização por enxame de partículas, o PSO. Em seguida, duas de suas extensões extraídas da literatura e utilizadas neste trabalho são descritas, enfatizando-se suas diferenças e vantagens em relação ao algoritmo original.

3.1. PSO

A afirmação “pensar é social” pode ser utilizada como motivador da utilização de um algoritmo como o PSO. Ela vem da ideia de que uma iteração grupal pode melhorar a capacidade cognitiva de um conjunto de indivíduos, que por sua vez aprendem e compartilham conhecimento com seus vizinhos. O assim denominado Modelo Cultural Adaptativo [Axe06] é baseado em três processos básicos: avaliar, comparar e imitar. Um sistema classifica estímulos em positivos ou negativos, define um referencial e aprende de acordo com esse mesmo referencial.

É nesse sentido que o PSO apresenta-se como uma técnica de otimização global baseada em uma população de soluções. De forma geral, o algoritmo é baseado no comportamento social de revoadas dos pássaros, onde um indivíduo imita as ações do melhor do grupo (ou mais indicado). O processo tem início com a definição da população de soluções. Cada indivíduo, por vezes chamado de “partícula”, é uma solução possível. Cada partícula possui uma posição e uma velocidade, e o processo de atualização é baseado na melhor experiência pessoal e na melhor experiência do grupo. O PSO foi criado por Kennedy e Eberhart em 1995 [KE95].

Seja *s* o tamanho do enxame, *n* a dimensão do problema e t o instante de tempo atual. Cada partícula $1 \leq i \leq s$ tem uma posição $x_i(t) \in \mathbb{R}^n$ no espaço da solução e uma velocidade $v_i(t) \in \mathbb{R}^n$, que por sua vez controla a magnitude, o sentido e a direção do movimento. Cada partícula mantém a melhor posição individual $y_i(t) \in \mathbb{R}^n$ visitada até o instante de tempo *t*. Por outro lado, o enxame como um todo mantém em memória a melhor posição $\hat{y}(t) \in \mathbb{R}^n$ visitada até agora por cada uma das partículas.

Ao longo do algoritmo, a velocidade de cada partícula é guiada por duas variáveis, ou pontos de busca: a melhor posição individual visitada até agora (termo cognitivo da otimização, $y_i(t)$) e a melhor posição global visitada até agora (termo social da otimização, $\hat{y}(t)$). Matematicamente, a nova velocidade de cada partícula é dada pela Equação 3.1, enquanto que a Equação 3.2 determina sua nova posição. O termo *j* indica a dimensão da partícula.

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_1 \left(y_{ij}(t) - x_{ij}(t)\right) + c_2 r_2 \left(\hat{y}_j(t) - x_{ij}(t)\right) \quad 1 \leq i \leq s, 1 \leq j \leq n \tag{3.1}$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \quad 1 \leq i \leq s, 1 \leq j \leq n \tag{3.2}$$

O chamado termo de *momentum* (ou inércia), *w*, provoca uma busca mais exploratória nas primeiras iterações e com nível maior de explotação nas últimas iterações. Essa variável é uma escalar que geralmente decresce linearmente de 0,9 a 0,4. As variáveis r_1 e r_2 são variáveis randômicas uniformemente variando de 0 a 1. Essas variáveis estão relacionadas com os dois termos da equação (cognitivo e social). c_1 e c_2 são os coeficientes de aceleração individual e local, respectivamente. Os coeficientes têm valores fixos e iguais, sendo responsáveis pelo controle do movimento da partícula em cada iteração.

A Equação 3.3 mostra como a melhor posição individual é atualizada, enquanto que a melhor posição global é atualizada de acordo com a Equação 3.4. É

importante frisar que a velocidade é determinada por um intervalo com os limites mínimo e máximo, evitando assim a “explosão” do enxame. O Algoritmo 1 apresenta o algoritmo do PSO padrão. *f* é a função de aptidão do algoritmo (ou *fitness*). Ela define se determinada posição de uma partícula tem desempenho melhor ou pior do que outra, determinando o curso de atualização das posições e das velocidades.

$$y_i(t+1) = \begin{cases} y_i(t), se\ f\big(x_i(t+1)\big) \geq f(y_i(t)) \\ x_i(t+1), se\ f\big(x_i(t+1)\big) < f(y_i(t)) \end{cases} \quad (3.3)$$

$$\hat{y}(t+1) = arg \min_{y_i(t+1)} f\big(y_i(t+1)\big), 1 \leq i \leq s \quad (3.4)$$

Algoritmo 1: PSO

iniciar aleatoriamente a população de partículas
repita
    para cada partícula i da população faça
        se $\boldsymbol{f}(\boldsymbol{x_i}(\boldsymbol{t})) < f(\boldsymbol{y_i}(\boldsymbol{t}))$ então
            $\boldsymbol{y_i}(\boldsymbol{t}) = \boldsymbol{x_i}(\boldsymbol{t})$
        fim se
        se $\boldsymbol{f}(\boldsymbol{y_i}(\boldsymbol{t})) < f(\hat{\boldsymbol{y}}(\boldsymbol{t}))$ então
            $\hat{\boldsymbol{y}}(\boldsymbol{t}) = \boldsymbol{y_i}(\boldsymbol{t})$
        fim se
    fim para
    atualizar velocidade e posição de cada partícula de acordo com eq. 3.1 e 3.2
até critério de parada ser satisfeito

Trabalhos como o de Van den Bergh [Van01], Clerc et al [CK02] e Trelea [Tre03] analisam matematicamente a convergência do PSO. Tais trabalhos tiveram como resultado orientações sobre quais parâmetros afetam a convergência, divergência ou oscilação do algoritmo, e esses estudos deram origem a diversas variações do PSO original. Entretanto, Pedersen [PC10] mostra que essas análises

são simplistas, no sentido em que assumem que o enxame possui somente uma partícula, que não utilizam variáveis estocásticas e que os pontos de atração permanecem constantes durante o processo de otimização. Adicionalmente, certas análises permitem um número infinito de iterações, impossível na realidade. Sendo assim, determinar as capacidades de convergência do PSO ainda depende de resultados empíricos.

3.2. Extensões do PSO

A seguir, serão apresentadas as extensões do PSO utilizadas neste trabalho.

3.2.1. EPUS-PSO

Embora o PSO padrão e diversas variações do algoritmo original tenham sido desenvolvidas e aplicadas em uma vasta gama de problemas de otimização, resolver problemas de alta complexidade com busca eficiente e rápida convergência tem sido um dos desafios nessa área de pesquisa. É nesse cenário que o EPUS-PSO (*Efficient Population Utilization Strategy for* PSO) [HSLT09] foi introduzido, buscando melhorar esses quesitos no algoritmo original. Para tanto, o EPUS-PSO se baseia em três conceitos, descritos a seguir.

Gerenciamento de população

Uma das inconveniências do algoritmo padrão do PSO é ter que decidir, a priori, o número de partículas do enxame, seja pela experiência do examinador ou pela complexidade do problema. Com um número grande de partículas, o espaço de busca aumenta juntamente com a probabilidade de encontrar um ótimo global, mas o tempo gasto com as iterações se torna maior. Por outro lado, um número pequeno de indivíduos que compõe o enxame pode restringir consideravelmente o espaço de busca, e o algoritmo pode ficar preso em um ótimo local.

Para reverter esse problema, o EPUS-PSO implementa o gerenciamento de população. Com esse conceito, o algoritmo tem a habilidade de aumentar ou diminuir o número de partículas de acordo com um determinado *status*, isto é, o tamanho da população é variável. O gerenciamento de população do EPUS-PSO ajusta o tamanho do enxame de acordo com três condições:

- Se $\hat{y}(t)$ (melhor posição global) não tiver sido atualizado em *k* iterações consecutivas e o tamanho da população não tiver excedido um limite pré-definido, uma nova partícula será adicionada ao enxame. Para que essa partícula seja posicionada em uma boa posição do enxame, ela será criada como uma combinação das melhores soluções passadas de duas partículas selecionadas randomicamente.
- Se $\hat{y}(t)$ tiver sido atualizado em *k* iterações consecutivas e o tamanho da população atual for maior que 1, a partícula com pior desempenho será removida. Essa ação considera que o tamanho do enxame é suficiente para um espaço de busca adequado e há indivíduos redundantes.
- Finalmente, se $\hat{y}(t)$ não for atualizado e o tamanho da população for igual a um valor máximo, a partícula com pior desempenho será removida para dar lugar a uma nova partícula com melhor desempenho em potencial.

Compartilhamento de solução

Na estratégia de compartilhamento de solução a equação de atualização de velocidade é dada pelas Equações 3.5:

$$\begin{gathered} v_{ij}(t+1) = \omega v_{ij}(t) + c_1 r_1 \left(y_{ij}(t) - x_{ij}(t)\right) + c_2 r_2 \left(\hat{y}_{ij}(t) - x_{ij}(t)\right) \\ \textit{se } rand \geq Ps_i \textit{ ou} \\ v_{ij}(t+1) = \omega v_{ij}(t) + c_1 r_1 \left(y_{ij}(t) - x_{ij}(t)\right) + c_2 r_2 \left(y_{rj}(t) - x_{ij}(t)\right) \\ \textit{caso contrário} \end{gathered} \tag{3.5}$$

O termo Ps_i diz respeito ao termo de compartilhamento de solução da i-ésima partícula. *rand* é uma função que retorna um valor real entre 0 e 1. Sendo $y_{rj}(t)$ a melhor posição local de uma outra partícula que não a i-ésima, essa nova expressão para atualização de velocidade faz com que a busca de soluções de todas as partículas utilize não somente as suas próprias melhores posições, mas também as melhores posições de outras partículas. Essa estratégia faz com que as partículas

possam adquirir conhecimento de outros indivíduos. Ps_i é determinada pela Equação 3.6. *N* denota a dimensão do problema e *s* o tamanho da população.

$$Ps_i = 0.5 * \frac{(N-1) * \exp\left(\frac{i-1}{s-1}\right) - 1}{2N} \tag{3.6}$$

***Searching Range Sharing* - SRS**

Essa estratégia é utilizada no EPUS-PSO no sentido de permitir a exploração em áreas que produzem mais soluções em potencial e encontrar espaço de soluções ainda não buscadas.

Segundo critérios randômicos, o SRS compartilha o *range* de busca tanto de partículas como de dimensões. A intenção principal é perturbar as soluções e abrir ainda mais o leque de possibilidades na busca. O critério randômico, $Pr(t)$, é determinado pela Equação 3.7.

$$Pr(t) = A_1 + \frac{A_2 * g}{max\_gen} \tag{3.7}$$

A_1 e A_2 são a taxa SRS inicial e a variação dessa taxa, respectivamente, geralmente dados por 0.03 e 0.07. O parâmetro *max_gen* é o número máximo de iterações, enquanto que *g* indica a geração da iteração atual.

A execução do algoritmo é bastante similar ao PSO padrão, com as devidas adaptações para considerar as estratégias utilizadas pelo EPUS-PSO com o intento de melhorar o desempenho da busca.

3.2.2. APSO

O *Adaptive Particle Swarm Optimization* (APSO) [ZZLC09] também foi desenvolvido com o objetivo de aumentar a eficiência da busca e da velocidade de convergência do algoritmo original do PSO. Basicamente, o APSO consiste de dois passos principais. Primeiro, o algoritmo identifica ao longo de sua execução, através da avaliação da distribuição da população e função de aptidão de cada partícula, em qual dos quatro estados evolucionários a geração de soluções se encontra:

exploração, explotação, convergência ou salto. Segundo, uma estratégia de aprendizado elitista é seguida, ativada quando o estado evolucionário é classificado como convergência.

O estado de exploração corresponde à busca inicial dos indivíduos, compreendendo as primeiras iterações. No estado de explotação, algumas partículas estão já agrupadas, possivelmente em ótimos locais. Já no estado de convergência, um ou mais grupos aproximam-se do ótimo global. Finalmente, no estado de salto, algumas partículas mudam sua localização no espaço de buscas, seguindo a partícula com melhor resultado. A noção de estados evolucionários foi incialmente introduzida em [ZCL07] e [ZXZC07]. Através do conhecimento desses estados evolucionários, há o controle automático de algumas variáveis do algoritmo, como o termo de inércia e os coeficientes de aceleração.

Para classificar o estado evolucionário da geração, o APSO utiliza a abordagem denominada ESE (*Evolutionary State Estimation*), baseada no comportamento de busca e distribuição da população de soluções do PSO. Tal abordagem pode ser resumida em 4 passos:

1. Na posição atual, calcule a distância média de cada partícula i para todas as outras partículas. A distância Euclidiana, segundo a Equação 3.8, pode ser utilizada (sendo *N* e *D* o tamanho da população e o número de dimensões, respectivamente).

$$d_i = \frac{1}{N-1} \sum_{j=1, j \neq i}^{N} \sqrt{\sum_{k=1}^{D} (x_i^k - x_j^k)^2} \tag{3.8}$$

2. Indique d_i da melhor partícula global como d_g. Compare todos os d_i'*s* e determine as distâncias mínimas (d_{min}) e máximas (d_{max}). O "fator evolucionário" *f* é dado pela Equação 3.9.

$$f = \frac{d_g - d_{min}}{d_{max} - d_{min}} \in [0, 1] \tag{3.9}$$

3. Classifique *f* em um dos quatros conjuntos S_1, S_2, S_3 e S_4, que representam os estados de exploração, explotação, convergência e salto. Uma

classificação do tipo *fuzzy* é recomendada [SE01]. A chave para essa classificação é sobrepor valores, segundo a implementação numérica descrita a seguir:

a. Caso (a) – Exploração

$$\mu S_1(f) = \begin{cases} 0, & 0 \leq f \leq 0.4 \\ (5 * f) - 2, & 0.4 < f \leq 0.6 \\ 1, & 0.6 < f \leq 0.7 \\ (-10 * f) + 8, & 0.7 < f \leq 0.8 \\ 0, & 0.8 < f \leq 1 \end{cases} \quad (3.10)$$

b. Caso (b) – Explotação

$$\mu S_2(f) = \begin{cases} 0, & 0 \leq f \leq 0.2 \\ (10 * f) - 2, & 0.2 < f \leq 0.3 \\ 1, & 0.3 < f \leq 0.4 \\ (-5 * f) + 3, & 0.4 < f \leq 0.6 \\ 0, & 0.6 < f \leq 1 \end{cases} \quad (3.11)$$

c. Caso (c) – Convergência

$$\mu S_3(f) = \begin{cases} 1, & 0 \leq f \leq 0.1 \\ (-5 * f) + 1.5, & 0.1 < f \leq 0.3 \\ 0, & 0.3 < f \leq 1 \end{cases} \quad (3.12)$$

d. Caso (d) – Salto

$$\mu S_4(f) = \begin{cases} 0, & 0 \leq f \leq 0.7 \\ (5 * f) - 3.5, & 0.7 < f \leq 0.9 \\ 1, & 0.9 < f \leq 1 \end{cases} \quad (3.13)$$

Em períodos de transição, *f* pode ser classificado como qualquer um dos estados sobrepostos, já que dois membros da função *fuzzy* podem ser ativados. Para obter uma classificação final, métodos de “desfuzificação” podem ser utilizados, como o *sigleton* e o *centroid* [JSM97]. No artigo original do APSO e neste trabalho o método *singleton* foi utilizado.

Tendo em vista como classificar o estado evolucionário de uma geração, a seguir serão apresentados os conceitos propostos pelo APSO com objetivo de melhorar o desempenho do PSO original.

Controle adaptativo dos parâmetros

1. Adaptação do termo de *momentum*

É sabido que o termo de *momentum* w, utilizado no PSO para balancear a capacidade de busca local e global, deve ser grande no estado de exploração e pequeno no estado de explotação [ES01] [SE98] [SE99]. Entretanto, somente diminuir w com o tempo pode não ser a solução ideal em todas as situações. Visto que o fator de evolução f apresenta comportamento semelhante com o termo de *momentum* em relação a seu tamanho na transição de estados, o APSO aproveita essa informação. Segundo a Equação 3.14, o termo de *momentum* segue os estados evolucionários utilizando um mapeamento sigmoidal.

$$w(f) = \frac{1}{1 + 1.5e^{-2.6f}} \in [0.4, 0.9] \quad \forall f \in [0, 1] \tag{3.14}$$

De forma geral, nos estados de exploração e salto, f e w são grandes e implicam em busca global. Por outro lado, tais variáveis decrescem nos estados de explotação e convergência para favorecer a busca local.

2. Controle dos coeficientes de aceleração

Como visto anteriormente, c_1 e c_2 são parâmetros que estão relacionados com a forma como a velocidade das partículas é atualizada. O primeiro diz respeito à influência da melhor posição da partícula sobre ela mesmo, ajudando-a a realizar busca local e manter a diversidade do enxame. Em contrapartida, c_2 representa a chamada “influência social”, por ponderar a melhor posição global. No APSO, os coeficientes de aceleração não possuem valor fixo. Eles são variáveis de acordo com o estado evolucionário, segundo as ações a seguir:

- Estado de exploração: aumentar c_1 e diminuir c_2.

- Estado de explotação: aumentar suavemente c_1 e diminuir suavemente c_2.
- Estado de convergência: aumentar suavemente ambos os parâmetros.
- Estado de salto: diminuir c_1 e aumentar c_2.

3. Limites dos coeficientes de aceleração

No APSO, os ajustes nos coeficientes de aceleração não são repentinos. Sua atualização é realizada de acordo com a equação 3.15.

$$|c_i(g+1) - c_i(g)| \leq \delta, \quad i = 1,2 \tag{3.15}$$

δ é a chamada taxa de aceleração com intervalo [0.05, 0.1]. Nas fases de explotação e convergência, com variações suaves dos coeficientes, a tal variável é atribuído o valor 0.05. Para evitar a "explosão" desses parâmetros, os coeficientes de aceleração são normalizados de acordo com a equação 3.16, quando sua soma é maior do que 4.0.

$$c_i = \frac{c_i}{c_i + c_2} 4.0, \quad i = 1,2 \tag{3.16}$$

***Elitist Learning Strategy* - ELS**

O objetivo principal do ELS no algoritmo do APSO é auxiliar a partícula com melhor desempenho global a sair de regiões de ótimos locais quando a busca estiver classificada como estado de convergência. Imprimir uma perturbação à partícula com melhor desempenho é importante porque esta não possui parâmetros para seguir. Se através dessa perturbação uma região com melhores resultados for encontrada, todo o enxame seguirá o exemplo da partícula líder.

O ELS seleciona aleatoriamente uma dimensão da partícula com melhor desempenho global e aplica uma perturbação gaussiana, segundo a equação 3.17 (sendo d a d-iésima dimensão da partícula e X os limites mínimos e máximos):

$$P^d = P^d + \left(X^d_{max} - X^d_{min}\right) \times Gaussian(\mu, \sigma^2) \tag{3.17}$$

Na distribuição gaussiana com média μ 0, o desvio-padrão (denominado taxa de aprendizado elitista no APSO) é dado pela equação 3.18. *g* indica a geração da iteração atual e *max_gen* denota o número máximo de gerações. σ é decrescido linearmente com o número de gerações.

$$\sigma = \sigma_{max} - (\sigma_{max} - \sigma_{min}) \frac{g}{\max\_gen} \tag{3.18}$$

Capítulo 4

Otimização de *Reservoir Computing*

Tendo em vista aumentar a eficiência de um sistema, inclusive os conexionistas, duas técnicas ou mais podem ser combinadas, caracterizando assim um Sistema Híbrido Inteligente (SHI). A ideia principal de um SHI é combinar duas ou mais técnicas com o objetivo de alcançar resultados melhores e mais robustos. Assim, busca-se mitigar as deficiências de determinada técnica e maximizar suas vantagens [BCL07].

A otimização de pelo menos uma das técnicas envolvidas em um SHI pode ser o objetivo principal da combinação [BCL07]. Segundo Panos e Douglass [PD00] a otimização é o processo de encontrar a melhor solução em um conjunto de possíveis soluções para um problema, sendo que o processo de busca parte de uma solução inicial (ou um conjunto delas) e realiza melhoramentos progressivos até chegar a uma solução ideal (ou um conjunto delas) dentro do espaço de busca. A otimização pode ser global ou local. Na otimização global, busca-se encontrar a melhor solução dentro de um conjunto de soluções em um espaço restrito. Por outro lado, na otimização local, tal busca depende do ponto de início do espaço de buscas. Nem sempre uma solução global é alcançada em termos práticos, pois o processo de busca pode ter um custo computacional maior. Em alguns cenários a solução local é eficiente o bastante [HT04].

A seguir serão apresentados trabalhos que tiveram o objetivo de otimizar RNA através de uma hibridização com outra técnica de computação inteligente. Em seguida, este mesmo conceito é discutido no campo de *Reservoir Computing*.

4.1. Otimização de Redes Neurais Artificiais

Diversos algoritmos podem ser utilizados para realizar otimização de RNA. Dentre estes, destacam-se Algoritmos Genéticos [Hol75], *Tabu Search* (TS) [GL97], *Simulated Annealing* (SA) [KGV83], PSO [KE95] e GSO (*Group Search Optimizer*) [HWS09]. A otimização pode estar relacionada com a arquitetura da rede, dos seus pesos e dos demais parâmetros livres.

Algoritmos Genéticos podem, por exemplo, otimizar ambos os parâmetros de configuração e arquitetura da rede [HSG89]. AG também podem ser utilizados para otimizar os valores iniciais das conexões entre as unidades de processamento [Köh96]. García-Pedrajas et al [GOH06] propõem novos operadores de recombinação para realizar essa tarefa. Mais recentemente, aplicações híbridas entre Algoritmos Genéticos e Redes Neurais Artificiais podem ser observadas em problemas do mundo real, como o aumento da eficiência em sistemas de e-commerce [SMR12] e classificação de imagens para auxílio na detecção de câncer de mama [WLM12].

Hamm [HBH02] utilizou *Simulated Annealing* para otimizar os pesos de uma Rede Neural Artificial. Metin et al [MSC+00] utilizaram essa mesma técnica para otimizar RNA no contexto de sistemas de diagnósticos auxiliados por computador.

Quanto à utilização de *Tabu Search* na tarefa de otimizar RNA, podem ser citados os trabalhos de Cannas et al [CFMP99] (*Tabu Search* para otimizar arquiteturas de redes neurais no campo de previsão de séries temporais) e Martí e El-Fallahi [ME04] (ajustes dos pesos de uma rede do tipo *Multi-Layer Perceptron*). *Tabu Search* também foi utilizado para otimizar redes neurais na tarefa de prever a força dos ventos [HLL11].

Combinar alguns desses algoritmos em um sistema híbrido pode alcançar resultados ainda melhores. É nesse sentido que Ludermir, Yamazaki e Zanchettin propuseram em 2006 uma técnica de otimização que combina SA e TS, na tarefa de otimizar Redes Neurais Artificiais [LYZ06]. A ideia principal desse algoritmo é, através da combinação de SA e TS, otimizar tanto os pesos quanto a arquitetura de uma rede neural do tipo MLP. A motivação para essa abordagem vem do fato de que a otimização simultânea tanto da arquitetura quanto dos pesos pode gerar redes

mais eficientes: cada ponto no espaço de busca é uma rede neural totalmente especificada, gerando assim uma função de custo mais confiável.

GaTSa é um acrônimo para *Tabu Search*, *Simulated Annealing* e *Genetic Algorithms*. Essas são as técnicas que, combinadas, servem como inspiração para um novo algoritmo de otimização, proposto por Zanchettin em 2008 [Zan08]. Assim como a técnica brevemente discutida no parágrafo anterior, o GaTSa otimiza simultaneamente tanto a topologia quanto os valores dos pesos das conexões em redes neurais do tipo MLP. A busca das soluções possíveis é feita de forma construtiva e também baseada na poda das conexões entre as unidades de processamento, gerando redes com arquitetura variável. Essa técnica de otimização pode ser utilizada na seleção de atributos principais, eliminando unidades de processamento segundo sua relevância para o desempenho do modelo.

O PSO também pode ser utilizado como metodologia para otimização global de redes neurais. É o que faz Carvalho [Car07] em sua dissertação de mestrado. O autor utilizou dois algoritmos PSO em redes do tipo MLP: um para a otimização de arquiteturas e outro para o ajuste dos pesos sinápticos de cada arquitetura gerada pelo primeiro PSO, sendo esses dois processos intercalados por um número específico de iterações. O trabalho gerou resultados satisfatórios, obtendo redes com baixo erro de generalização e baixa complexidade. Lima e Ludermir utilizaram o Frankenstein PSO também para otimizar arquitetura e pesos de uma rede neural *feedforward* [LL11]. O Frankenstein PSO é uma extensão do PSO original baseado em diversas outras modificações. O PSO também foi utilizado como método de seleção de atributos em redes neurais desse tipo, apresentando resultados eficientes [JJQ12].

O uso do PSO na otimização de redes neurais pode ser observado em aplicações recentes do mundo real: classificação de imagens de retina [AVSH12], roteamento de redes *ad-hoc* [SH12] e otimização de condições de cultura de uma bactéria utilizada na indústria farmacêutica [KSHB12].

O GSO é um algoritmo de enxame inteligente baseado no comportamento social de pesquisa dos animais e na teoria de vida em grupo. A utilização do GSO no âmbito da otimização de RNA já é difundida na literatura. He et al utilizaram GSO para treinar uma rede neural aplicada ao diagnóstico de câncer de mama [HWS09]. Essa mesma arquitetura foi utilizada em [HL08] para monitoramento da condição de

máquinas de ultrassom. Silva e Ludermir utilizaram os chamados GSO cooperativos no sentido de aumentar o poder de generalização das redes [SL11]. Os GSO cooperativos mostraram desempenho maior do que o algoritmo tradicional nas bases de dados investigadas.

4.2. Otimização de *Reservoir Computing*

Como visto anteriormente, as arquiteturas primordiais de *Reservoir Computing* (LSM e ESN) trabalham com uma rede neural recorrente com pesos fixos, gerados aleatoriamente. Entretanto, Lukosevicius e Jaeger [LJ09] indicam que é improvável que a geração aleatória dos pesos e o treinamento da camada de saída com uma função de regressão linear simples seja a solução ideal para computar RC, sendo necessárias pesquisas de alternativas para geração do *reservoir* e treinamento do *readout*.

Como alternativa para a geração da camada de *reservoir*, é possível criar os pesos da camada intermediária segundo adaptações não supervisionadas ou mesmo a partir de um pré-treinamento supervisionado. A adaptação não supervisionada pressupõe otimizar alguma medida definida no reservoir ($x(n)$) para uma dada entrada ($u(n)$), não levando em consideração a saída desejada ($y_{target}(n)$). Em contrapartida, o pré-treinamento supervisionado leva também em consideração a saída desejada.

Na adaptação não supervisionada, diversas medidas podem ser utilizadas para estimar a qualidade de um *reservoir*. Lukosevicious e Jaeger citam a minimização da faixa dos autovalores (EVS – *Eigenvalue Spread*) [LJ09]. O EVS é a relação entre os valores máximo e mínimo dos autovalores da matriz de correlação cruzada das ativações $x(n)$. Os mesmo autores indicam que a entropia da distribuição de $x(n)$ também pode ser utilizada como fator para otimização, buscando maximizá-la. Outra medida é a chamada “beira do caos” [LM07]. A “beira do caos” de um sistema dinâmico é a região onde seus parâmetros operam na fronteira entre o regime estável e o regime caótico. Entretanto, calcular esse fator não é trivial e muitas vezes requer um especialista.

Em relação à adaptação supervisionada no processo de geração do *reservoir*, metodologia essa considerada otimização de RC, alguns trabalhos podem ser citados. Ishiiet al [IvdZBP04] e Bush e Tsendjav [BT05] realizaram uma abordagem

evolucionária para otimizar *reservoirs*. O primeiro trabalho utilizou uma busca evolucionária dos parâmetros de geração da matriz de pesos do *reservoir* (número de neurônios no *reservoir*, raio espectral e densidade das conexões), objetivando criar um sistema para movimentação de um robô submarino. Também foi testada a busca dos pesos da matriz de *reservoir* com menos neurônios, tendo em vista diminuir o espaço de busca. Segundo os autores, os resultados foram superiores a outros métodos. Bush e Tsendjav também realizaram uma pesquisa evolucionária da topologia da rede, alcançando erros na previsão de comportamento de um sistema de amortecedor 50% menores do que a computação realizada sem qualquer tipo de otimização.

Além da adaptação do *reservoir*, outras maneiras para otimizar *Reservoir Computing* podem ser consideradas. Ferreira e Ludermir [FL09] apresentaram um método para otimizar a escolha dos parâmetros globais utilizando Algoritmos Genéticos, aplicando o sistema a um problema real de previsão de séries temporais. A otimização levou até 22,22% do tempo necessário para realizar uma busca exaustiva dos parâmetros e alcançar resultados semelhantes. Na busca exaustiva, todos os parâmetros são combinados sem utilizar qualquer método de otimização.

Em 2011, Ferreira [Fer11] desenvolveu um método para encontrar o melhor *reservoir* aplicado à tarefa de prever séries temporais, denominado RCDESIGN. O método busca simultaneamente os melhores valores dos parâmetros globais, da topologia da rede e dos pesos, através da combinação de um algoritmo evolucionário com *Reservoir Computing*. Outros dois métodos de otimização foram implementados para realizar a comparação dos resultados. A chamada Busca RS tenta otimizar o tamanho do *reservoir*, o raio espectral e a densidade de conexão. A Busca TR busca simultaneamente o raio espectral e a topologia da rede. Juntamente com o RCDESIGN, a Busca TR não considera a aproximação de RC com Sistemas Lineares (não reescala a matriz de pesos do *reservoir* pelo raio espectral e permite criação de sistemas com as conexões opcionais de RC). De um modo geral, o RCDESIGN apresentou resultados satisfatórios em todas as bases de dados estudadas, sendo melhor que os outros métodos utilizados para comparação. Todas as abordagens foram também aplicadas à previsão da velocidade dos ventos, que é uma tarefa importante para geração de energia eólica.

O PSO tem propriedades e aplicações semelhantes aos Algoritmos Genéticos. Ambos são populacionais, utilizam operadores sobre o conjunto de soluções em um determinado processo iterativo e são inspirados em comportamentos da natureza. Entretanto, o PSO possui a vantagem de ter implementação mais simples e, em alguns casos, convergência relativamente mais rápida e custo computacional menor [HCdWV05] [PP07].

Tendo em vista essas características, o PSO é um importante candidato para realizar otimização de *Reservoir Computing*. Além disso, as extensões desse algoritmo podem aumentar ainda mais a eficiência da tarefa, visto que esse conceito já foi aplicado com sucesso na literatura (por exemplo, o GSO cooperativo apresentou resultados melhores do que o GSO original na otimização de redes *feedforward*). Sergio e Ludermir utilizaram o PSO e duas de suas extensões (EPUS-PSO e APSO) para otimizar os parâmetros globais do RC, concluindo que nas cinco séries temporais utilizadas o método proposto reduziu o número de ciclos de treinamento necessários para treinar o sistema [SL12]. Os parâmetros utilizados na otimização foram o número de nodos no *reservoir*, função de ativação dos neurônios do *reservoir*, raio espectral da matriz de pesos do *reservoir* e presença ou ausência das conexões opcionais.

Em resumo, na tarefa de previsão de séries temporais, otimizar os parâmetros globais e os pesos de *Reservoir Computing* pode trazer resultados satisfatórios, como visto com o RCDESIGN [Fer11]. Além disso, substituir Algoritmos Genéticos pelo PSO pode implicar em aumento de desempenho e computação mais rápida. Resultados preliminares com o PSO indicam sua eficiência [SL12]. Este trabalho busca dar continuidade à pesquisa iniciada em [SL12]: baseando-se no RCDESIGN, a dissertação propõe um método para otimizar os parâmetros globais, a topologia e os pesos de *Reservoir Computing*, utilizando o PSO e duas de suas extensões.

Capítulo 5

Otimização de *Reservoir Computing* com PSO

Este capítulo tem o objetivo de apresentar o método utilizado, incluindo a descrição do algoritmo aplicado nas simulações numéricas. Em seguida, o método experimental é apresentado, incluindo as bases de dados e a descrição da ferramenta utilizada para a realização dessas simulações.

5.1. Otimização de *Reservoir Computing* com PSO

O método utilizado será apresentado a seguir. Este trabalho tem o objetivo de aplicar o PSO na tarefa de buscar os parâmetros globais, a topologia da rede e os pesos de um *Reservoir Computing* na previsão de séries temporais. Será descrito como as soluções foram representadas, de que forma a função de aptidão foi calculada, o algoritmo de otimização utilizado e os parâmetros envolvidos. A otimização é adaptada do método proposto por Ferreira, o RCDESIGN [Fer11]. Ao invés de Algoritmos Genéticos, o PSO foi utilizado como algoritmo de otimização. O ESN foi utilizado como método de *Reservoir Computing*.

5.1.1. Representação das Soluções

No PSO, existem os conceitos de partícula (uma solução individual) e população de partículas (um conjunto de soluções ou enxame). Ao utilizar esse algoritmo, é necessário definir a representação dessa solução.

A representação utilizada por Ferreira [Fer11] foi adaptada para a inclusão do PSO. Cada partícula é representada por um vetor s^i. A notação s_j^i denota a dimensão *j* da partícula s^i. A seguir, cada uma dessas dimensões é descrita.

- s_1^i (η) – Número de nodos no *reservoir*, inteiro entre 50 e 200. Esse parâmetro define o tamanho das matrizes de pesos. Ferreira [Fer11] indica que essa amplitude de valores constrói *reservoirs* com tamanho suficiente para obtenção de bons resultados e não grande o bastante para que o processo seja demorado.
- s_2^i – Conexão entre a camada de entrada e a camada de saída. Número real entre 0 e 1. Se maior que 0.5, há conexão, se menor, não há.
- s_3^i – Conexão entre o *bias* e a camada de saída. Número real entre 0 e 1. Se maior que 0.5, há conexão, se menor, não há.
- s_4^i – Conexão de realimentação na camada de saída. Número real entre 0 e 1. Se maior que 0.5, há conexão, se menor, não há.
- s_5^i – Conexão entre o *bias* e a camada de *reservoir*. Número real entre 0 e 1. Se maior que 0.5, há conexão, se menor, não há.
- s_6^i – Conexão entre a camada de saída e a camada de *reservoir*. Número real entre 0 e 1. Se maior que 0.5, há conexão, se menor, não há.
- s_7^i – Função de ativação dos neurônios. Se 1, tangente hiperbólica, se 2, sigmóide.
- s_8^i – Função de treinamento do *readout*. Se 1, *pseudo-inversa* [LJ09], se 2, *ridge-regress* [Bis06].
- s_9^i – *Leak rate*. Número real entre 0,1 e 1.
- s_{10}^i – Parâmetro de regularização das funções de treinamento. Número real entre 10^{-8} e 10^{-1}.
- s_{11}^i ... $s_{(\eta^2+3\eta+10)}^i$ – Pesos das camadas de *reservoir* (W), entrada (W^{in}), *bias* (W^{bias}) e realimentação (W^{back}). Número real entre 0,1 e 1.

O tamanho do vetor s^i é variável. Isso acontece porque as últimas posições dependem do número de nodos no *reservoir*, definido pela dimensão *j* = 1. O intervalo [50; 200] foi definido empiricamente, com a intenção de criar *reservoirs* grande o suficiente para computar os dados e com tamanho viável para realizar as

simulações. Se η = 50, o vetor s^i tem 2660 posições. Caso η = 200, o vetor s^i tem tamanho máximo de 40610. A divisão do vetor levando em consideração os pesos do *reservoir* pode ser observada na Figura 7.

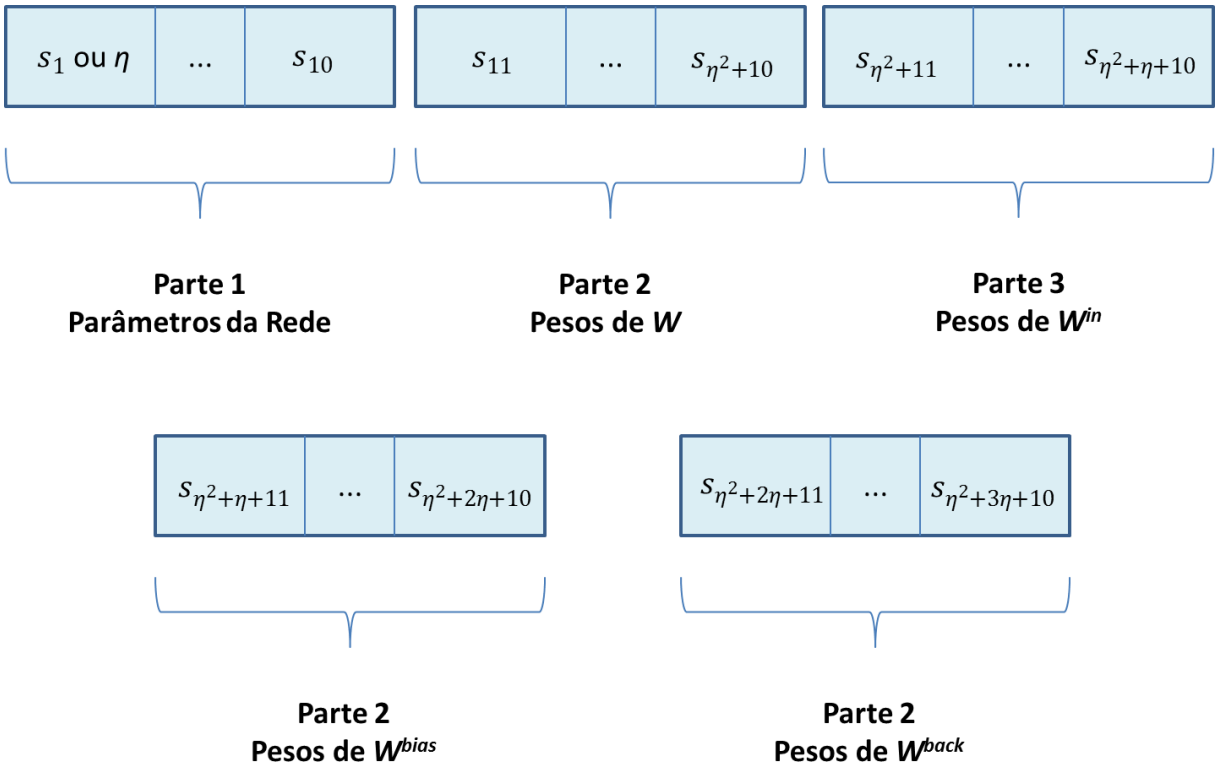

Figura 7. Divisão conceitual de s^i.

Leak rate é um parâmetro que pode ser adicionado aos neurônios, possibilitando o ajuste da dinâmica de RC. Se o *leak rate* é escolhido adequadamente, a dinâmica do sistema pode ser ajustada para coincidir com a escala de tempo da entrada, resultando em melhor desempenho [ASS08]. O parâmetro de regularização corresponde a um ruído que pode ser adicionado às respostas do *reservoir*.

5.1.2. Função de Aptidão

Para que o algoritmo do PSO guie as partículas para os valores ótimos no espaço de busca, é necessário definir uma boa função de aptidão, ou *fitness*. A função de aptidão resulta em um valor que representa o desempenho de determinada partícula

no enxame. O objetivo do algoritmo é maximizar ou minimizar esse valor, dependendo do que a função representa.

Na otimização de *Reservoir Computing* com PSO, foi utilizada como medida de desempenho a função de aptidão do RCDESIGN [Fer11]. Essa função de aptidão é baseada no erro médio quadrático dos sistemas criados (MSE, do inglês *Mean Square Error*). Devido ao fenômeno de *overfitting*, comum em arquiteturas de Redes Neurais Artificiais, ela leva em consideração não somente o MSE do sistema criado no conjunto de treinamento, mas também no de validação. A equação 5.1 mostra a função de *fitness* utilizada. O MSE é calculado de acordo com a equação 5.2.

$$f = \overline{MSE}_{Treinamento} + \left\| \overline{MSE}_{Treinamento} - \overline{MSE}_{Validação} \right\| \tag{5.1}$$

$$MSE = \frac{1}{N * P} \sum_{i=1}^{P} \sum_{j=1}^{N} (T_{ij} - L_{ij})^2 \tag{5.2}$$

Na equação 5.2, *P* é o número de padrões no conjunto de dados, *N* é o número de unidades de saída da rede e T_{ij} e L_{ij} são respectivamente o valor desejado de saída e o valor calculado pelo *i-ésimo* neurônio da camada de saída.

5.1.3. Algoritmo de Otimização

O algoritmo da otimização de *Reservoir Computing* com PSO é apresentado a seguir, no Algoritmo 2. Assim como no caso da função de aptidão, o algoritmo é baseado no RCDESIGN [Fer11]. É importante salientar que neste trabalho, para cada base de dados, três simulações numéricas são realizadas. Uma usando o algoritmo padrão do PSO e outras duas utilizando as extensões selecionadas: EPUS-PSO e APSO.

O treinamento das redes foi realizado com o método de validação cruzada (*k-fold cross-validation*). Na utilização do método da validação cruzada, o número de partições é um parâmetro importante a ser definido. Um valor pequeno obtém resultados mais rápidos, porém com erros maiores. Por outro lado, um valor muito grande pode demorar a apresentar resultados, em contrapartida a apresentar erros

menores. A validação cruzada com 10 partições tem se mostrado um valor adequado para a maioria dos problemas [WF00].

Algoritmo 2: Otimização de Reservoir Computing com PSO

Entrada: Base de Dados, Tamanho do Enxame = *s*, Número de Iterações = *iterMax*, Abordagem (PSO, EPUS-PSO ou APSO)
Selecione Base de Dados
Selecione Abordagem
Inicialize enxame de tamanho *s* com partículas posicionadas aleatoriamente no espaço de busca
enquanto (iteração ≤ *iterMax*)
    Crie RC de acordo com posição da partícula
    enquanto *fold* ≤ 10 (validação cruzada) **faça**
        Crie conjuntos de treinamento (nove partições) e conjunto de validação (uma partição)
        Simule rede com conjunto de treinamento
        Treine pesos da camada linear de saída (*readout*)
        Calcule erros no conjunto de treinamento
    fim enquanto
    Calcule função de aptidão
    Atualize posição e velocidade das partículas de acordo com Abordagem
fim enquanto
Retorne a melhor solução
Crie RC de acordo com melhor solução
Calcule os erros no conjunto de teste

5.1.4. Parâmetros

No PSO, bem como em suas extensões, diversos parâmetros devem ser definidos. Os valores desses parâmetros representam um importante fator para o desempenho do algoritmo. Entretanto, similarmente como ocorre na utilização de redes neurais, não há regras claras sobre quais valores esses parâmetros devem assumir.

Os parâmetros do PSO neste trabalho foram definidos segundo processo empírico, de tentativa e erro. Diversos conjuntos de parâmetros foram testados aleatoriamente, de acordo com o bom senso do projetista. Os testes também levaram em consideração valores de parâmetros que são consagrados na literatura, como o termo de *momentum* inicial. Foi selecionada a configuração de parâmetros que apresentou melhor desempenho nos testes iniciais. Abaixo, os parâmetros utilizados.

- Tamanho do enxame: 50;
- Número de iterações: 20;
- Termo de *momentum* inicial: 0.9;
- Coeficientes de aceleração global e local: 2.05.

No caso específico do algoritmo do EPUS-PSO, o tamanho do enxame e o número de iterações são variáveis. Nesse caso, os valores iniciais para esses parâmetros foram 60 e 20, respectivamente.

Outro ponto importante é a amplitude dos valores dos pesos. Foi detectado empiricamente que em algumas bases de dados os resultados eram melhores com um intervalo menor do que [-1; 1]. Contudo, esses continuaram a serem os valores máximos para todas as bases.

5.2. Método Experimental

A seguir serão descritas as bases de dados utilizadas nas simulações numéricas, seguidas das abordagens para comparação entre a otimização proposta e outros métodos presentes na literatura.

5.2.1. Bases de Dados

Foram utilizadas sete séries temporais clássicas utilizadas como *benchmark* na literatura e três bases com aplicação prática, contendo dados da velocidade média horária dos ventos na região Nordeste do Brasil.

Séries Narma ordem 10 e Narma ordem 30

A série NARMA (*Nonlinear Autoregressive Moving Average*) é uma série discreta regida pela equação 5.3:

$$(y+1) = 0.3y(t) + 0.05y(t)\left[\sum_{i=0}^{k-1} y(t-i)\right] + 1.5u\big(t-(k-1)\big) * u(t) + 0.1 \quad (5.3)$$

A entrada da série é um ruído aleatório uniforme *u(t)*, *t* é o tempo, *k* é a ordem do sistema e *y* é a saída. Foram utilizados dois valores para a ordem do sistema: *k = 10* e *k = 30*. De um modo geral, as séries Narma são difíceis devido a sua não linearidade e necessidade de uma memória longa.

Séries *Mackey-Glass* caos médio e *Mackey-Glass* caos moderado

A série *Mackey-Glass*, contínua, unidimensional e *benchmark* padrão para teste de previsão de séries temporais é formada pela equação 5.4:

$$y(t+1) = \frac{0.2y(t-\tau)}{1+y(t-\tau)^{10}} - 0.1y(t) \quad (5.4)$$

y(t) é a saída no tempo *t* e τ é um parâmetro de atraso que influencia o nível de caos da série. Foram utilizados dois valores para o parâmetro τ: τ*= 17* (caos médio) e τ*=30* (caos moderado). Em ambos os casos o sistema é considerado caótico, sendo que 17 é menos caótico que 30.

***Multiple Sinewave Oscillator* (MSO)**

A série MSO é utilizada para criar um sistema de geração de múltiplos sinos. Sendo d(t) o valor de sinal no tempo t, a série é gerada pela equação 5.5.

$$d(t) = \sin(0.2 * t) + \sin(0.311 * t) \quad (5.5)$$

Série natural do Brilho da Estrela (STAR)

Disponível em http://robjhyndman.com/TSDL/physics/, a série STAR é composta por 600 observações numéricas sucessivas da luminosidade à meia noite.

Série financeira (*Dow Jones Industrial Average*)

Disponível em http://www.djindexes.com/, a série financeira *Dow Jones Industiral Average* consiste das observações diárias do índice de mesmo nome. Os dados utilizados neste trabalho contêm 1444 registros, com observações no período de 02 de Janeiro de 1998 a 26 de Agosto de 2003.

Velocidade Média Horária dos Ventos na Região Nordeste do Brasil

A geração eólica é, dentre outras variáveis, uma função da velocidade dos ventos. Prever adequadamente esses valores pode reduzir sensivelmente as dificuldades de operação em uma fonte eólica e também em fontes tradicionais de energia, já que é possível integrar modelos eficientes de previsões de geração de energia eólica com outros sistemas elétricos de geração [AJL+10].

As três séries utilizadas no trabalho de Ferreira [Fer11] foram selecionadas, obtidas no site do projeto SONDA (Sistema de Organização de Dados Ambientais) [end13a]. O projeto SONDA mantém bases de dados gratuitas dos recursos de energia solar e eólica do Brasil. São elas:

- Série de Belo Jardim: Município de Pernambuco. Série com 13176 padrões constituída pelas velocidades médias horárias dos ventos obtidas pela central eólica de Belo Jardim, de 01 de Julho de 2004 a 31 de Dezembro de 2005.
- Série de São João do Cariri: Município da Paraíba. Série com 17520 padrões constituída pelas velocidades médias horárias dos ventos obtidas pela central eólica de São João do Cariri, de 01 de Janeiro de 2006 a 31 de Dezembro de 2007.
- Série de Triunfo: Município de Pernambuco. Série com 21936 padrões constituída pelas velocidades médias horárias dos ventos obtidas pela central eólica de Triunfo, de 01 de Julho de 2004 a 31 de Dezembro de 2006.

5.2.2. Abordagens para Comparação

No intuito de comparar os resultados obtidos entre si e com outros métodos na literatura, o desempenho da otimização proposta neste trabalho foi calculado segundo diversos índices de erros de previsão. São eles: erro médio quadrático (MSE), raiz quadrada do erro médio quadrático normalizado (NRMSE, do inglês *Normalized Root Mean Square Error*), erro médio quadrático normalizado (NMSE, do inglês *Normalized Mean Square Error*), e nos casos das bases de vento percentagem de erro absoluto (MAPE, do inglês *Mean Absolute Percentage Error*). As medidas são especificadas respectivamente pelas equações 5.2, 5.6, 5.7 e 5.8.

$$NRMSE = \frac{1}{N * P} \sum_{i=1}^{P} \sum_{j=1}^{N} sqrt(\frac{\left(T_{ij} - L_{ij}\right)^2}{var(t)}) \tag{5.6}$$

$$NMSE = \frac{1}{N * P} \sum_{i=1}^{P} \sum_{j=1}^{N} \frac{\left(T_{ij} - L_{ij}\right)^2}{var(t)} \tag{5.7}$$

$$MAPE = 100 * \frac{1}{N * P} \sum_{i=1}^{P} \sum_{j=1}^{N} |\frac{T_{ij} - L_{ij}}{T_{ij}}| \tag{5.8}$$

Nestas equações, *P* é o número de padrões no conjunto de dados, *N* é o número de unidades de saída da rede e T_{ij} e L_{ij} são respectivamente o valor desejado de saída e o valor calculado pelo *i-ésimo* neurônio da camada de saída. *var(T)* é a variância dos valores no conjunto de saídas desejadas. Tendo em vista que o MSE, o NMSE e o NRMSE elevam o erro ao quadrado, maiores erros penalizam a avaliação final mais acentuadamente. O MAPE calcula a medida de precisão da previsão realizada pelo sistema.

Para dar significância estatística à comparação dos resultados, foi utilizado o teste de hipóteses *t de Student*. Testes de hipóteses são procedimentos estatísticos que calculam a probabilidade de alguma propriedade relacionada aos dados serem

verdadeiras ou não, a partir de determinadas pressuposições (hipótese nula). O teste *t de Student* rejeita ou não a hipótese nula quando a estatística de teste aproxima-se de uma distribuição t de *Student*, isto é, simétrica e semelhante à curva normal padrão, com caudas mais longas. Neste trabalho, foram realizados testes não pareados e bilaterais (testa se os valores são estatisticamente iguais) entre os resultados alcançados, a partir de 30 amostras e com significância de 0.05. Mais detalhes sobre testes estatísticos podem ser encontrados em [Wai07].

5.2.3. Toolbox de RC no Matlab

Para facilitar a condução dessa pesquisa, foi utilizado um *toolbox* de *Reservoir Computing* no Matlab (RCT – *Reservoir Computing Toolbox*) [SVH12]. O *toolbox* oferece um conjunto de funções e *scripts* com o intuito de facilitar experimentos com RC. O RCT permite ao usuário criar topologias para o *reservoir*, manipular parâmetros do experimento e processar os resultados. O código é *open source* sob licença GPL (*General Public License*). A Figura 8 mostra o fluxo geral de simulação do RCT. A ilustração auxilia o entendimento do processo de treinamento e utilização de um *Reservoir Computing*.

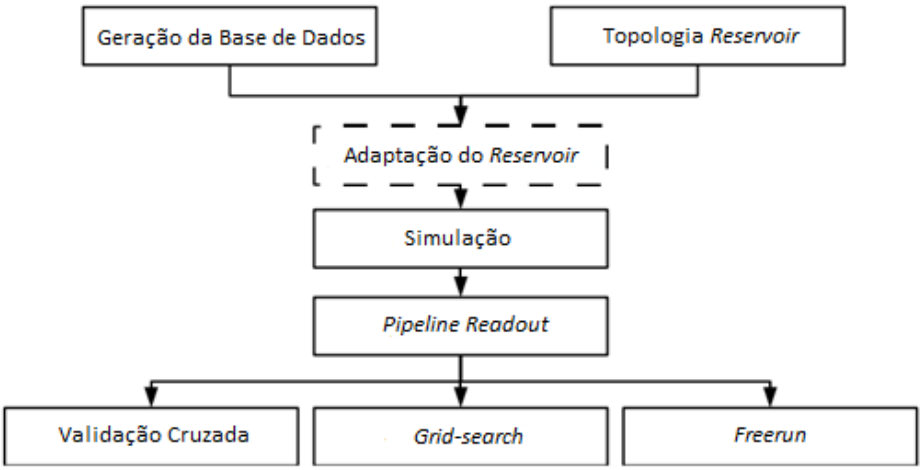

Figura 8. Fluxo geral de simulação do RCT

Usualmente, um experimento consiste da geração de uma base de dados e da construção de uma camada *reservoir* de determinada topologia. O *reservoir*, então, pode ser pré-adaptado utilizando a fase de adaptação. A simulação é depois realizada e os resultados do *reservoir* são salvos. Finalmente, na fase *readout*

pipeline, uma função é treinada e o desempenho do RC pode ser avaliado através de três metodologias distintas: validação cruzada, *grid-search* (busca exaustiva do espaço de parâmetros) ou *freerun* (o *readout* é treinado com todos os exemplos da base da dados, no caso de séries temporais). Na fase de adaptação, opcional, o *reservoir* é simulado alimentando o sistema com entradas em ordem randômica, enquanto certos parâmetros internos da rede são treinados em modo não supervisionado.

No RCT, a topologia do sistema pode ser definida graficamente de acordo com a matriz mostrada na Figura 9. O *toolbox* permite conectar entradas, *reservoir* e saídas tanto com o *reservoir* quanto com as saídas. Diversas configurações são permitidas, exceto quando da manipulação da conexão do *reservoir* com a saída (treinada baseada na base de dados). No caso mais simples, há somente a conexão entre entrada e *reservoir*, a conexão de feedback no *reservoir* e a conexão entre o *reservoir* e a camada de saída.

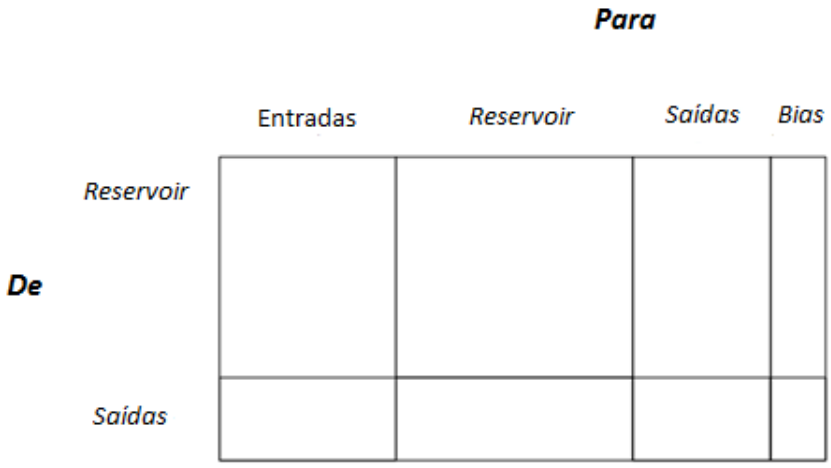

Figura 9. Topologia no RCT

O PSO e suas extensões foram implementados separadamente e o algoritmo de otimização foi posteriormente integrado ao RCT. Diversos *toolboxes* abertos de PSO podem ser encontrados na Internet [end13b] [end13c], mas houve a preferência da implementação direta por esta opção apresentar maior flexibilidade.

Capítulo 6

Simulações Numéricas

As simulações numéricas serão apresentadas neste capitulo. Os resultados no treinamento e nos testes serão descritos de acordo com diversas medidas de desempenho. Em seguida, os resultados alcançados serão comparados com outros trabalhos na literatura.

6.1. Resultados

Nas tabelas 1, 2 e 3 são apresentados respectivamente o MSE, o NMSE e o NRMSE resultantes dos três métodos propostos na fase de treinamento. Devido às características aleatórias das simulações numéricas, e para realização de teste de hipóteses, as medidas de desempenho estão representadas pelas médias de 30 inicializações de cada base de dados. Os valores entre parêntesis são os desvios-padrão. Em destaque, os melhores desempenhos.

A tabela 4 mostra a comparação entre os métodos de acordo com o teste *t* de *Student* não pareado ao nível 5% de significância (95% de confiança), em relação ao NRMSE. Nessa tabela, o sinal "=" indica que a hipótese nula não foi rejeitada (a diferença entre as médias dos erros não é estatisticamente relevante) e os modelos apresentam o mesmo desempenho. O sinal "<" indica que a hipótese nula foi rejeitada e que os modelos para comparação (segundo modelo na comparação) tem desempenho inferior ao de seleção (primeiro modelo na comparação). Por fim, o sinal ">" indica que a hipótese nula foi rejeitada e que os modelos para comparação tem desempenho superior ao de seleção. O NRMSE foi escolhido para realização do teste de hipóteses por possuir valores com menos casas decimais, facilitando os cálculos.

Tabela 1. MSE no treinamento, 30 inicializações.

	PSO	EPUS-PSO	APSO
Narma Ordem 10	0,0000288660 (0,0000)	0,0000446050 (0,0000)	0,0000332154 (0,0000)
Narma Ordem 30	0,0001681646 (0,0001)	0,0001863113 (0,0001)	0,0001028620 (0,0000)
MGS Médio	0,0000000005 (0,0000)	0,0000000007 (0,0000)	0,0000000005 (0,0000)
MGS Moderado	0,0000000091 (0,0000)	0,0000000098 (0,0000)	0,0000000089 (0,0000)
MSO	0,0000000000 (0,0000)	0,0000000000 (0,0000)	0,0000000000 (0,0000)
STAR	0,0000728987 (0,0000)	0,0000735891 (0,0000)	0,0000694666 (0,0000)
Down-Jones	0,0007744125 (0,0000)	0,0007751246 (0,0000)	0.0007747537 (0,0000)
BJD	0,0049285697 (0,0000)	0,0049557216 (0,0000)	0,0049408270 (0,0000)
SCR	0,0035703998 (0,0000)	0,0035805862 (0,0000)	0,0035628694 (0,0000)
TRI	0,0008909650 (0,0000)	0,0008928967 (0,0000)	0,0008849037 (0,0000)

Tabela 2. NMSE no treinamento, 30 inicializações.

	PSO	EPUS-PSO	APSO
Narma Ordem 10	0,0024442115 (0,0016)	0,0037451683 (0,0060)	0,0027539714 (0,0039)
Narma Ordem 30	0,0140995225 (0,0084)	0,0156043271 (0,0097)	0,0086606500 (0,0036)
MGS Médio	0,0000000123 (0,0000)	0,0000000158 (0,0000)	0,0000000120 (0,0000)
MGS Moderado	0,0000001370 (0,0000)	0,0000001467 (0,0000)	0,0000001334 (0,0000)
MSO	0,0000000000 (0,0000)	0,0000000000 (0,0000)	0,0000000000 (0,0000)
STAR	0,0010741993 (0,0001)	0,0010843578 (0,0000)	0,0010236145 (0,0001)
Down-Jones	0,0174703413 (0,0001)	0,0174858090 (0,0000)	0,0174767444 (0,0000)
BJD	0,2540956111 (0,0028)	0,2554954649 (0,0025)	0,2547276267 (0,0029)
SCR	0,1603587736 (0,0019)	0,1608165898 (0,0016)	0,1600204821 (0,0013)
TRI	0,0481392157 (0,0021)	0,0481051150 (0,0009)	0,0476743853 (0,0013)

É possível observar que, no treinamento, apesar de o APSO ter alcançado melhor desempenho em termos absolutos, os métodos são estatisticamente semelhantes entre si na maioria das bases de dados estudadas. A ressalva fica por conta do EPUS-PSO, que obteve desempenho inferior ao PSO na base MSO.

As tabelas 5, 6 e 7 mostram o desempenho na fase de testes dos três métodos propostos, de acordo com o MSE, o NMSE e o NRMSE. Assim como na

fase de treinamento, os valores foram coletados a partir da média de 30 inicializações. Em destaque, os melhores desempenhos.

A tabela 8 mostra o teste *t* de *Student* para comparação estatística dos NRMSE alcançados pelos três métodos na fase de teste. A configuração do teste *t* e a lógica da tabela é a mesma que foi apresentada na fase de treinamento.

De acordo com a tabela 8, é possível observar que o EPUS-PSO não superou o desempenho do PSO e do APSO em nenhuma base de dados. O APSO superou o PSO em três bases, sendo que nas outras sete alcançou resultados estatisticamente similares de acordo com o teste de hipóteses realizado. Em relação ao EPUS-PSO, o APSO teve desempenho superior em oito bases de dados, enquanto o PSO superou o EPUS-PSO em sete.

Tabela 3. NRMSE no treinamento, 30 inicializações.

	PSO	EPUS-PSO	APSO
Narma Ordem 10	0,0476193864 (0,0135)	0,0536939439 (0,0298)	0,0477106861 (0,0222)
Narma Ordem 30	0,1139923173 (0,0337)	0,1199777505 (0,0353)	0,0910574919 (0,0195)
MGS Médio	0,0001100644 (0,00001)	0,0001246725 (0,00001)	0,0001082409 (0,00001)
MGS Moderado	0,0003691350 (0,00002)	0,0003821484 (0,00002)	0,0003643903 (0,00002)
MSO	0,0000000038 ($9*10^{-10}$)	0,0000000044 ($9*10^{-9}$)	0,0000000033 ($6*10^{-9}$)
STAR	0,0326679960 (0,0021)	0,0327247176 (0,0033)	0,0318399628 (0,0027)
Down-Jones	0,1317907032 (0,0001)	0,1318504343 (0,00001)	0,1318174990 (0,0001)
BJD	0,5040350481 (0,0027)	0,5054228162 (0,0025)	0,5046604367 (0,0029)
SCR	0,4004123502 (0,0023)	0,4009851492 (0,0021)	0,3999931857 (0,0016)
TRI	0,2191917770 (0,0048)	0,2191550209 (0,0022)	0,2181637190 (0,0028)

Tabela 4. Comparação estatística no treinamento – NRMSE.

	N10	N30	MGS17	MGS30	MSO	STAR	DJ	BJD	SCR	TRI
PSO x EPUSPSO	=	=	=	=	=	=	<	=	=	=
PSO x APSO	=	=	=	=	=	=	=	=	=	=
EPUSPSO x APSO	=	=	=	=	=	=	=	=	=	=

Tabela 5. MSE no teste, 30 inicializações.

	PSO	EPUS-PSO	APSO
Narma Ordem 10	0,0000317203 (0,0000)	0,0000497007 (0,0000)	0,0000372386 (0,0000)
Narma Ordem 30	0,0001799486 (0,0001)	0,0002002863 (0,0001)	0,0001109147 (0,0000)
MGS Médio	0,0000000006 (0,0000)	0,0000000008 (0,0000)	0,0000000006 (0,0000)
MGS Moderado	0,0000000098 (0,0000)	0,0000000105 (0,0000)	0,0000000096 (0,0000)
MSO	0,0000000000 (0,0000)	0,0000000000 (0,0000)	0,0000000000 (0,0000)
STAR	0,0001112137 (0,0000)	0,0001140024 (0,0000)	0,0001085345 (0,0000)
Down-Jones	0,0007795072 (0,0000)	0,0007807437 (0,0000)	0,0007793334 (0,0000)
BJD	0,0051267509 (0,0000)	0,0051375377 (0,0000)	0,0051251231 (0,0000)
SCR	0,0037418917 (0,0000)	0,0037587109 (0,0000)	0,0037371017 (0,0000)
TRI	0,0009041416 (0,0000)	0,0009057100 (0,0000)	0,0008958765 (0,0000)

Tabela 6. NMSE no teste, 30 inicializações.

	PSO	EPUS-PSO	APSO
Narma Ordem 10	0,0027065080 (0,0017)	0,0041075755 (0,0064)	0,0031188580 (0,0046)
Narma Ordem 30	0,0155248435 (0,0094)	0,0172223773 (0,0107)	0,0094620809 (0,0041)
MGS Médio	0,0000000128 (0,0000)	0,0000000165 (0,0000)	0,0000000124 (0,0000)
MGS Moderado	0,0000001484 (0,0000)	0,0000001590 (0,0000)	0,0000001443 (0,0000)
MSO	0,0000000000 (0,0000)	0,0000000000 (0,0000)	0,0000000000 (0,0000)
STAR	0,0030128323 (0,0001)	0,0030738498 (0,0001)	0,0029324917 (0,0001)
Down-Jones	0,1513301644 (0,0007)	0,1519443288 (0,0005)	0,1516449084 (0,0005)
BJD	0,3070486821 (0,0006)	0,3076181756 (0,0007)	0,3069249956 (0,0008)
SCR	0,1953642874 (0,0010)	0,1962510815 (0,0011)	0,1951531745 (0,0010)
TRI	0,1472917843 (0,0061)	0,1474227093 (0,0031)	0,1459830180 (0,0037)

Utilizando os erros de previsão na fase de teste como medida de desempenho para comparação das abordagens propostas, pode-se concluir que o APSO alcançou os melhores resultados. É possível que a estratégia ESE (*Evolutionary State Estimation*), utilizada no APSO, tenha se adequado melhor às características dos dados estudados. No ESE, o algoritmo do PSO tenta identificar

em quais dos quatro diferentes estados evolucionários (exploração, explotação, convergência e salto) sua execução se encontra, para poder assim calcular as variáveis necessárias de diferentes maneiras (termo de *momentum*, coeficientes de aceleração e os limites desses coeficientes). A natureza das séries temporais, onde a ordem dos dados é relevante, diferentemente dos modelos de regressão linear, pode ter oferecido um padrão melhor aplicável ao ESE.

Tabela 7. NRMSE no teste, 30 inicializações.

	PSO	EPUS-PSO	APSO
Narma Ordem 10	0,0499853761 (0,0141)	0,0562621742 (0,0301)	0,0503082103 (0,0242)
Narma Ordem 30	0,1193474167 (0,0353)	0,1257601918 (0,0371)	0,0948630262 (0,0208)
MGS Médio	0,0001119798 (0,00001)	0,0001270548 (0,00001)	0,0001099572 (0,00001)
MGS Moderado	0,0003829910 (0,00002)	0,0003960918 (0,00002)	0,0003778040 (0,00002)
MSO	0,0000000039 ($8*10^{-10}$)	0,0000000045 ($1*10^{-9}$)	0,0000000034 ($7*10^{-10}$)
STAR	0,0501224563 (0,00001)	0,0506885487 (0,0010)	0,0494933321 (0,0001)
Down-Jones	0,3780419257 (0,0008)	0,3787736114 (0,0006)	0,3783935519 (0,0006)
BJD	0,5513885351 (0,0006)	0,5519262169 (0,0006)	0,5512637857 (0,0007)
SCR	0,4387079294 (0,0011)	0,4397166635 (0,0012)	0,4384478650 (0,0011)
TRI	0,3723567172 (0,0075)	0,3726854490 (0,0039)	0,3707453994 (0,0045)

Tabela 8. Comparação estatística no teste – NRMSE.

	N10	N30	MGS17	MGS30	MSO	STAR	DJ	BJD	SCR	TRI
PSO x EPUSPSO	=	=	<	<	<	<	<	<	<	=
PSO x APSO	=	>	=	=	>	>	=	=	=	=
EPUSPSO x APSO	=	>	>	>	>	>	>	>	>	=

Uma explicação para o desempenho inferior do EPUS-PSO seria o fato deste algoritmo possuir um problema da mesma natureza cujo trabalho em questão tenta solucionar. No EPUS-PSO, assim como na configuração de experimentos com redes neurais artificiais, é necessário ao projetista definir o valor inicial de certos parâmetros sem qualquer conhecimento prévio do problema. Nesta extensão do PSO, o número de partículas varia ao longo da execução do algoritmo, necessitando

da definição inicial de parâmetros que controlem essa dinâmica e do valor inicial para o tamanho do enxame. Tais parâmetros foram definidos com base nas orientações do artigo que propôs a extensão, mas é provável que para cada tipo de problema o direcionamento possa ser distinto.

Na execução do algoritmo de otimização de *Reservoir Computing* proposto, seja com a utilização do PSO ou com uma de suas extensões, a tarefa com maior custo computacional é o cálculo da função de aptidão. É nesse ponto que o algoritmo realiza um ciclo completo de treinamento do *reservoir*, para calcular o MSE na fase de treinamento e validação. Logo, o número de vezes que a função de aptidão é calculada está intimamente ligado com o custo computacional do algoritmo como um todo. Uma solução mais eficiente segundo esse ponto de vista busca realizar menos ciclos de treinamento possíveis.

No PSO, o número de ciclos de treinamento (que é igual ao número de cálculos da função de aptidão), é imutável em cada execução e dado pela multiplicação entre o tamanho do enxame e o número de iterações do algoritmo. Nas simulações realizadas, esse número é dado por 50 (tamanho do enxame) * 20 (iterações) = 1000. No caso do APSO, o raciocínio é similar, com a diferença de que são realizados cálculos a mais da função de aptidão para a fase ELS do algoritmo. Esse número é indefinido, visto que o fluxo ELS do algoritmo é somente executado quando a otimização está na fase de convergência.

Já no caso do EPUS-PSO, o tamanho do enxame e o número de iterações são variáveis ao longo da execução do algoritmo. Dessa forma, o número de ciclos de treinamento do *reservoir* é também variável. A tabela 9 mostra a média dos ciclos de treinamento realizados pelo EPUS-PSO para cada uma das bases de dados.

Tais resultados mostram que a utilização do EPUS-PSO obteve sempre os melhores resultados segundo o critério de ciclos de treinamento necessários para alcançar os valores ótimos. Quando em determinada aplicação o tempo de execução é mais crítico do que o desempenho do algoritmo propriamente dito, o EPUS-PSO é mais indicado. O desempenho do EPUS-PSO pode ser ainda melhorado investigando-se uma melhor configuração de parâmetros e valores iniciais para o tamanho inicial do enxame e o número de iterações.

Tabela 9. Média dos ciclos de treinamento realizados no EPUS-PSO.

Base de Dados	Média dos Ciclos de Treinamento
Narma Ordem 10	807,07
Narma Ordem 30	871,67
MGS Médio	858,07
MGS Moderado	869,90
MSO	877,87
STAR	861,97
Down-Jones	864,50
BJD	863,27
SCR	867,83
TRI	850,60

6.2. Comparação com Outros Trabalhos

Como explicitado anteriormente, vários trabalhos na literatura tiveram por objetivo otimizar *Reservoir Computing* na resolução de diversos tipos de problemas. Esta sessão tem o objetivo de resgatar os resultados alcançados por esses trabalhos e compará-los com a pesquisa descrita nesta dissertação.

Ferreira [Fer11] utilizou abordagem semelhante ao método proposto. A autora utilizou Algoritmos Genéticos para buscar parâmetros, arquitetura e pesos do *reservoir*. Foram investigadas três abordagens, o RCDESIGN, a Busca RS e a Busca TR. Apesar dos experimentos de Ferreira não terem sidos reproduzidos nesse trabalho, a sua configuração é bastante similar às simulações descritas na sessão anterior. Assim, com as devidas restrições, os resultados podem ser comparados e discutidos. A tabela 10 compara os NRMSE das abordagens de Ferreira com o APSO do método proposto. O APSO foi selecionado para realizar essa comparação por ter alcançando o melhor desempenho de acordo com o erro de previsão. Os valores mostram a média de 30 execuções e os desvios-padrão entre parêntesis. O melhor desempenho para cada base de dados está destacado.

A tabela 11 mostra o teste *t* de *Student* não pareado ao nível 5% de significância (95% de confiança) para comparação estatística dos NRMSE

alcançados pelos três métodos de Ferreira e pelo APSO. A configuração do teste *t* e a lógica da tabela são similares às outras apresentadas nesse capítulo.

Tabela 10. Comparação do NRMSE no teste.

	APSO	RCDESIGN	Busca RS	Busca TS
Narma 10	0,0503082103 (0,0242)	0,08462155 (0,0354)	0,42436505 (0,0438)	0,20807155 (0,0246)
Narma 30	0,0948630262 (0,0208)	0,20424126 (0,0469)	0,41465769 (0,0460)	0,20796290 (0,0180)
MGS 17	0,0001099572 (0,00001)	0,00050628 (0,0001)	0,00584469 (0,0006)	0,00168976 (0,0002)
MGS 30	0,0003778040 (0,00002)	0,00099067 (0,0002)	0,01114130 (0,0020)	0,00260901 (0,0004)
MSO	0,0000000034 ($7*10^{-10}$)	0,00000120 (0,0000)	0,01347762 (0,0021)	0,00068574 (0,0002)
STAR	0,0494933321 (0,0001)	0,08555901 (0,2088)	0,07059365 (0,0044)	0,05330699 (0,0020)
Down-Jones	0,3783935519 (0,0006)	0,23938190 (0,0156)	0,30542720 (0,0045)	0,23371477 (0,0021)
BJD	0,5512637857 (0,0007)	0,49230377 (0,0025)	0,83375926 (0,0039)	0,52168618 (0,0040)
SCR	0,4384478650 (0,0011)	0,40928555 (0,0030)	0,78283935 (0,0052)	0,45727073 (0,0062)
TRI	0,3707453994 (0,0045)	0,43600947 (0,0016)	0,82799176 (0,0103)	0,45196280 (0,0048)

Através da tabela 11, é possível observar que a otimização com o APSO foi melhor do que a Busca RS em todas as bases de dados estudadas, exceto *Down-Jones*. Em relação à Busca TS, o APSO foi superado em duas bases, a série de velocidade dos ventos de Belo Jardim (BJD) e *Down-Jones*. O melhor desempenho do APSO em relação à Busca RS e à Busca TS pode vir do fato de que a otimização com o APSO possui um espaço de busca mais versátil, com a maior possibilidade de encontrar uma melhor configuração de parâmetros adequada ao problema.

Tabela 11. Comparação estatística no teste – NRMSE.

	N10	N30	MGS17	MGS30	MSO	STAR	DJ	BJD	SCR	TRI
APSO x RCDESIGN	<	<	<	<	<	=	>	>	>	<
APSO x Busca RS	<	<	<	<	<	<	>	<	<	<
APSO x Busca TS	<	<	<	<	<	<	>	>	<	<

A otimização de RC com APSO (bem como com as outras abordagens propostas) possui configuração bastante similar ao RCDESIGN. Por exemplo, ambos atuam sobre o mesmo espaço de busca e utilizam a mesma função de aptidão. A diferença fica por conta do algoritmo de otimização utilizado: PSO ou Algoritmos Genéticos. O teste de hipóteses apresentado na tabela 11 mostra que, em relação ao NRMSE, o APSO foi melhor do que o RCDESIGN em seis das bases de dados investigadas. Obteve desempenho similar em uma base (*STAR*) e foi superado pelo RCDESIGN em três.

Outra medida de desempenho que pode ser utilizada para comparar os resultados alcançados pela otimização com PSO e o RCDESIGN é o critério de ciclos de treinamento necessários para alcançar os valores ótimos. Como visto anteriormente, segundo esse critério o EPUS-PSO alcançou os melhores resultados, tendo em vista que foi o que necessitou de menos cálculos da função de aptidão, nunca passando de 900 (vide tabela 9). O RCDESIGN, nas bases de dados estudadas, calcula a função de aptidão em cada uma delas no mínimo 1200 vezes, visto que o tamanho de população nas simulações é 120 e o número de gerações é 10 (120 * 10 = 1200). Na base *STAR*, por exemplo, onde os resultados foram estatisticamente semelhantes, é possível inferir que a abordagem com PSO convergiu mais rápido do que o RCDESIGN. Mesmo o APSO possui menos ciclos de treinamento do que o RCDESIGN.

A tabela 12 mostra o MAPE no teste para o APSO e as abordagens de Ferreira [Fer11], para as bases de velocidade média horária dos ventos. Com

resultado similar à comparação do NRMSE, o APSO teve desempenho superior em uma base de dados, da cidade de Triunfo (TRI).

Tabela 12. MAPE no teste, 30 inicializações.

	APSO	RCDESIGN	Busca RS	Busca TR
BJD	17,8384 (0,06)	12,08 (0,11)	20,27 (0,11)	12,47 (0,07)
SCR	17,3678 (0,06)	13,29 (0,28)	24,75 (0,43)	13,70 (0,51)
TRI	8,5348 (0,09)	9,86 (0,04)	20,63 (0,24)	10,23 (0,09)

Embora a configuração das simulações seja diferente, é possível comparar o desempenho do APSO com outros trabalhos que utilizaram alguma das bases de dados estudadas, com as restrições decorrentes da não reprodução de seus experimentos. Jaeger [Jae3] e Steil [Ste04] tentaram otimizar a base Narma 10. O primeiro utilizou uma abordagem baseada no algoritmo *Recursive Least Squares* (RLS) e o segundo realizou experimentos com o algoritmo *Atyia-Parlos Recurrent Learning* (APRL) e o *Backpropagation-Decorrelation* (BPCD). Steil também realizou experimentos com a base Mackey-Glass caos médio.

Verstraeten et al [VSDS07] realizaram um estudo sobre a relação entre os parâmetros do RC e sua dinâmica, utilizando a base Narma 10 e concluindo que o tamanho da rede não era importante, mas sim a função de ativação e o raio spectral. Boedecker et al [BOMA09] utilizaram Narma 30 para investigar um método para inicialização da matriz de pesos da camada intermediária, usando permutação de matrizes. Também para Narma 30, Schrauwen et al [SWV+08] utilizaram uma adaptação da regra *Intrinsic Plasticity*.

Jaeger [Jae01], Jaeger e Haaas [JH04] e Wyffels et al [WSVS08] realizaram experimentos de RC para Mackey-Glass caos médio e moderado. Para o MSO, Schmidhuber et al [SWGG07] realizaram simulações com o método EVOLINO (*EVOlution of recurrent systems with LINear Outputs*).

Sergio e Ludermir [SL12] buscaram otimizar Narma 10, Narma 30, Mackey-Glass caos médio e Mackey-Glass caos moderado também com PSO. Entretanto, o algoritmo só buscou parâmetros da rede, não os pesos do *reservoir*.

A tabela 13 compara o APSO com o trabalho de Sergio e Ludermir [SL12], que utilizou o MSE para quantificar o desempenho. A tabela 14 mostra a

comparação com os trabalhos que utilizaram o NMSE como medida de desempenho, enquanto a tabela 15 mostra a comparação com os trabalhos que utilizaram o NRMSE. O APSO obteve melhor desempenho em todas as bases de dados, quando levado em consideração os erros de previsão.

Tabela 13. Comparação do MSE.

	MSE
Narma 10	
APSO	0,000037
Sergio e Ludermir [SL12]	0,000162
Narma 30	
APSO	0,000110
Sergio e Ludermir [SL12]	0,000131
MGS Médio	
APSO	0,0000000006
Sergio e Ludermir [SL12]	0,0000394980
MGS Moderado	
APSO	0,0000000096
Sergio e Ludermir [SL12]	0,0000255480

Tabela 14. Comparação do NMSE.

	NMSE
Narma 10	
APSO	0,0031
Jaeger [Jae03]	0,0081
Steil [Ste04]	0,1420
MGS Médio	
APSO	0,0000000124
Steil [Ste04]	0,0340

Os resultados mostram que a otimização de *Reservoir Computing* com PSO, bem como com as suas extensões selecionadas, apresentaram desempenho

satisfatório para todas as bases de dados estudadas. A otimização superou o desempenho de diversos trabalhos na literatura, apresentando-se como uma solução importante para o tipo de problema investigado, a previsão de séries temporais.

Tabela 15. Comparação do NRMSE.

	NRMSE
Narma 10	
APSO	0,050
Verstraeten et al [VSDS07]	0,400
Narma 30	
APSO	0,0948
Boedecker et al [BOMA09]	0,4542
Schrauwen et al [SWV+08]	0,4600
MGS Médio	
APSO	0,00010
Jaeger [Jae01]	0,00012
Jaeger e Haas [JH04]	0,000063
Wyffels et al [WSVS08]	0,0065
MGS Moderado	
APSO	0,00037
Jaeger [Jae01]	0,032
Wyffels et al [WSVS08]	0,0065
MSO	
APSO	0,0000000034
Schmidhuber et al [SWGG07]	0,0103

Capítulo 7

Conclusões e Trabalhos Futuros

Este capítulo apresentará as conclusões do trabalho e as propostas de trabalhos futuros.

7.1. Conclusões

Este trabalho apresentou um método para otimizar simultaneamente os parâmetros globais, a topologia e os pesos de *Reservoir Computing* utilizando o PSO. A otimização foi realizada com o algoritmo padrão do PSO e duas de suas extensões presentes na literatura, o EPUS-PSO e o APSO.

A metodologia desenvolvida consistiu na adaptação do método proposto por Ferreira [Fer11] no problema de previsão de séries temporais, utilizando o PSO. Foram utilizadas sete bases de dados de *benchmark* e três séries temporais contendo os dados de velocidade dos ventos de cidades do Nordeste brasileiro, informação essa importante para a geração de energia eólica.

Para comparar as diferentes abordagens (PSO, EPUS-PSO e APSO), foram utilizadas quatro diferentes índices para os erros de previsão: o MSE, o NMSE, o NRMSE e o MAPE. Testes estatísticos foram realizados na comparação dos resultados. Outra abordagem para comparação dos métodos foi o número de ciclos de treinamento do *reservoir*, visto que essa informação influencia diretamente na velocidade da solução.

O APSO foi o algoritmo que apresentou os melhores resultados segundo os erros de previsão. O EPUS-PSO obteve os melhores resultados quando o critério de

ciclos de treinamento necessários para alcançar os valores ótimos foi levado em consideração.

Os resultados alcançados foram comparados com outros trabalhos na literatura. De acordo com os erros de previsão, a otimização com o APSO mostrou-se melhor do que as outras otimizações na maioria das bases de dados investigadas. Entretanto, é importante observar que as simulações numéricas dos trabalhos utilizados para comparação não foram reproduzidos. Tal restrição deve ser levada em consideração.

De um modo geral, a otimização de *Reservoir Computing* com PSO (seja com o algoritmo padrão ou com as extensões selecionadas) mostrou-se eficiente em todas as bases de dados estudadas. Os resultados obtidos, de acordo com as abordagens utilizadas, fazem com que o método proposto mostre-se como uma solução importante na literatura no que tange a tarefa de otimizar RC.

Importante salientar que essa dissertação gerou a publicação de um artigo na *International Conference on Artificial Neural Networks*, edição 2012: *PSO for Reservoir Computing Optimization* [SL12].

7.2. Trabalhos Futuros

A seguir, alguns temas para pesquisas futuras relacionadas a esse trabalho.

- **Utilizar outras extensões do PSO**. As simulações numéricas mostraram que, mesmo partindo do algoritmo original, as extensões do PSO podem seguir caminhos diferentes no processo de otimização e apresentar resultados estatisticamente distintos. É possível que haja na literatura alguma extensão do PSO que se adeque melhor ao problema tratado.
- **Utilizar outros algoritmos de otimização**. O método proposto utilizou o PSO, mas também já foram estudadas outros algoritmos de otimização (o método utilizado foi adaptado do proposto por Ferreira [Fer11], no qual a autora sugere a aplicação de outros algoritmos). Apesar disso, diversas outras técnicas ainda não foram investigadas em *Reservoir Computing*, como ACO (*Ant Colony Optimization* – Otimização por Colônia de Formigas) e GSO (*Group Searh Optimizer*). Adicionalmente, as mais diversas técnicas podem ser combinadas entre si e apresentar melhores resultados no problema de otimizar RC.

- **Aplicar o método proposto em outras bases de dados de aplicação prática**. A otimização de RC com PSO foi aplicada em três bases de dados de previsão da velocidade dos ventos em três cidades do Nordeste brasileiro. Seria interessante investigar o uso desse método de otimização em outras bases de uso prático. É possível inclusive adaptar a solução para aplicação em bases de dados de classificação.

Referências

[AJL+10] R. R. B. Aquino, M. A. Carvalho Junior, M. M S. Lira, O. Nóbrega Neto, G. J. Almeida, S. N. N. Tiburcio. *Recurrent neural networks solving a real large scale mid-term scheduling for power plants*. In International Joint Conference on Neural Networks - IJCNN 2010, pp. 3439–3444, Barcelona, 2010.

[Ami89] D. J. Amit. *Modeling brain function: The world of attractor neural networks*. New York, NY: Cambridge University Press, 1989.

[AS07] A. Atakulreka, D. Sutivong. *Avoiding Local Minima in Feedforward Neural Networks by Simultaneous Learning*. Australian Joint Conference on Artificial Intelligence - AUS-AI, pp. 100-109, 2007.

[ASS08] E. A. Antonelo, B. Schrauwen, D. Stroobandt. *Event detection and localization for small mobile robots using reservoir computing*. Neural Networks, 21(6), pp. 862–871, 2008.

[AVSH12] J. Anitha, K. S. Vijila, I. A. Selvakunar, J. D. Hemanth. *Performance enhanced PSO-based modified Kohonen neural network for retinal image classification*. Journal of the Chinese Institute of Engineers, 35(8), pp. 979-991, 2012.

[Axe06] R. Axelrod. *The Evolution of Cooperation*. Basic Books, 2006.

[BCL07] A. P. Braga, A. P. L. F. Carvalho, T. B. Ludermir. *Redes Neurais Artificiais: Teoria e Aplicações*. ETC, 2° Ed., 2007.

[BHOM09] J. Bersin,C. Howard, K.O'Leonard, D. Mallon. *Learning Management Systems*. Bersin & Associates, 2009.

[Bis06] C. M. Bishop. Pattern recognition and machine learning. In Information Science and Statistics. Springer, 2006.

[BM95] D. V. Buonomano, M. M. Merzenich. *Temporal information transformed into spatial code by a neural network with realistic properties*. Science, 267, pp. 1028-1030, 1995.

[BOMA09] J. Boedecker, O. Obst, N. M. Mayer, M. Asada. *Studies on reservoir initialization and dynamics shaping in echo state networks*. 17th European Symposium on Artificial Neural Networks, 2009.

[BSF94] Y. Bengio, P. Simard, P. Frasconi. *Learning long-term dependencies with*

gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2), pp. 157-166, 1994.

[BT05] K. Bush, B. Tsendjav. *Improving the richness of echo state features using next ascent local search*. Artificial Neural Networks In Engineering Conference, pp. 227–232, 2005.

[Bur05a] H. Burgsteiner. *On learning with recurrent spiking neural networks and their applications to robot control with real-world devices*. Tese de Doutorado, Graz University of Technology, 2005.

[Bur05b] H. Burgsteiner. *Training networks of biological realistic spiking neurons for real-time robot control*. 9th international conference on engineering applications of neural networks, pp. 129–136, 2005.

[BVV+11] P. Buteneers, D. Verstraeten, P. van Mierlo, T.Wyckhuys, D.Stroobandt, R. Raedt, H. Hallez, B. Schrauwen. *Automatic detection of epileptic seizures on the intra-cranial electroencephalogram of rats using reservoir computing*. Artificial Intelligence in Medicine, 53, pp. 215-223, 2001.

[Car07] M. R. Carvalho. *Uma Análise da Otimização de Redes Neurais MLP por Enxames de Partículas*. Dissertação de Mestrado. Universidade Federal de Pernambuco, CIn, Ciência da Computação, 2007.

[CFMP99] B. Cannas, A. Fanni, M. Marchesi, F. Pilo. *A Tabu Search algorithm for optimal sizing of locally recurrent neural networks*. 10th International Symposium On Theoretical Electrical Engineering, Magdeburg, Germany, pp. 267-272, 1999.

[CK02] M. Clerc, J. Kennedy. *The particle swarm - explosion, stability, and convergence in a multidimensional complex space*. IEEE Transactions on Evolutionary Computation, 6(1), pp. 58–73, 2002.

[DAJ95] P. Dominey, M. Arbib, J. P. Joseph. *A model of corticostriatal plasticity for learning oculomotor association and sequences*. J. Cogn. Neurosci., 7(3), pp. 311-336, 1995.

[Elm90] J. L. Elman. *Finding Structure in Time*. Cognitive Science, 14, pp. 179-211, 1990.

[end13a] SONDA, Sistema de Organização Nacional de Dados Ambientais. http://sonda.cptec.inpe.br/, Janeiro 2013.

[end13b] PSO Toolbox. http://psotoolbox.sourceforge.net/, Janeiro 2013.

[end13c] PSO Research Toolbox. http://www.georgeevers.org/pso_research_toolbox.htm, Janeiro 2013.

[ES01] R. C. Eberhart, Y. H. Shi. *Particle swarm optimization: Developments, applications and resources*. Proc. IEEE Congr. Evol. Comput., Seoul, Korea, pp. 81–86, 2001.

[Fah91] S. E. Fahman. *The Recurrent Cascade-Correlation Architecture*. Technical Report CMU-CS-91-100, Carnegie-Mellon University, 1991.

[Fer11] A. Ferreira. *Um Método para Design e Treinamento de Reservoir Computing Aplicado à Previsão de Séries Temporais*. Tese de Doutorado, Universidade Federal de Pernambuco, CIn, Ciência da Computação, 2011.

[FL08] A. A. Ferreira, T. B. Ludermir. *Using reservoir computing for fore-casting time series: Brazilian case study*. International Conference on Hybrid Intelligent Systems - HIS 2008, pp. 602–607, Los Alamitos, CA, USA, 2008.

[FL09] A. A. Ferreira, T. B. Ludermir. *Genetic algorithm for reservoir computing optimization*. International Joint Conference on Neural Networks, pp. 811-815, 2009.

[GL97] F. Glover, M. Laguna. *Tabu Search*. Kluwer academic publishers, Boston, 1997.

[GOH06] N. García-Pedrajas, D. Ortiz-Boyer, C. Hervás-Martínez. *An alternative approach for neural network evolution with a genetic algorithm: crossover by combinatorial optimization*. Neural Networks, 19, pp. 514-528, 2006.

[GMC+92] C. L. Gilesl, C. B. Miller, D. Chen, H.H. Chen, G. Z. Sun, Y. C. Lee. *Learning and extracting finite state automata with second-order recurrent neural networks*. Neural Computation, 4, pp. 393-405, 1992.

[GWM00] A. Gupta, Y. Wang, H.Markram. *Organizing principles for a diversity of GABAergic interneurons and synapses in the neocortex*. Science 287, pp. 273-278, 2000.

[Hay99] S. Haykin. *Neural Networks: a Comprehensive Foundation*. Prentice Hall, 2° Ed., 1999.

[HBH02] L. Hamm, B. W. Brorsen, M. T. Hagan. *Global optimization of neural network weights*. Proceedings of the 2002 International Joint Conference on Neural Networks, 2, pp. 1228-1233, 2002.

[HCdWV05] R. Hassan, B. Cohanim, O. De Weck, G. Venter. *A Comparison Of Particle Swarm Optimization And The Genetic Algorithm*. 46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, pp. 1-13, 2005.

[HL08] S. He, X. Li. *Application of a Group Search Optimization based Artificial*

Neural Network to Machine Condition Monitoring. 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2008), pp. 1260-1266, 2008.

[HLL11] S. Han, JS. Li, YQ Liu. *Tabu Search Algorithm Optimized ANN Model for Wind Power Prediction with NWP*. Proceedings of International Conference on Smart Grid and Clean Energy, 12, 2011.

[Hol75] Holland, H.J. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[HSG89] S. A. Harp, T. Samad, A. Guha. *Towards the genetic synthesis of neural networks*. Proc. of the third Int'l Conf. on Genetic Algorithms and Their Applications (J. D. Schaffer, ed.), Morgan Kaufmann, San Mateo, CA, pp. 360-369, 1989.

[HSLT09] S. Hsieh, T. Sun, C. Liu, S. J. Tsai. *Efficient Population Utilization Strategy for Particle Swarm Optimizer*. IEEE Transactions, Man and Cybernetics, 30, pp. 444-456, 2009.

[HT04] H. H. Hoos, T. Stützle. *Stochastic local search: foundations and applications*. Morgan Kaufmann / Elsevier, 2004.

[HWS09] S. He, H. Wu, J. R. Saunders. *Group Search Optimizer: An Optimization Algorithm Inspired by Animal Searching Behaviour*. IEEE Transactions on Evolutionary Computation, 13(5), pp. 973-990, 2009.

[Hyo96] H. Hyoetyniemi. *Turing machines are recurrent neural networks*. Proc. of SteP'96 –Genes, Nets and Symbols, pp. 13-24, 1996.

[IvdZBP04] K. Ishii, T. van der Zant, V. Becanovic, P. Ploger. *Identification of motion with echo state network*. Ocean 2004 Conference, 3, pp. 1205–1210, 2004.

[Jae01] H. Jaeger. *The echo state approach to analyzing and training recurrent neural networks*. Tech. Rep. GMD 148, German National Resource Center for Information Technology, 2001.

[Jae02] H. Jaeger. *Short-term memory in echo state networks*. Technical report, GDM 152, German National Resource Center for Information Technology, 2002.

[Jae03] H. Jaeger. *Adaptive nonlinear system identification with echo state networks*. Advances in Neural Information Processing Systems, 15, pp. 593–600, 2003.

[Jae05] H. Jaeger. *Reservoir riddles: Suggestions for echo state network research*. International Joint Conference on Neural Networks - IJCNN 2005, 3, pp.

1460–1462, 2005.

[Jae07] H. Jaeger. *Discovering multiscale dynamical features with hierarchical echo state networks*. Technical report 10, School of Engineering and Science, Jacobs University, 2007.

[JH04] H. Jaeger, H. Haas. *Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless telecommunication*. Science, 308, pp. 78–80, 2004.

[JJQ12] C. Jin, SW. Jin, LN. Qin. *Attribute selection method based on a hybrid BPNN and PSO algorithms*. Applied Soft Computing, 12(8), pp. 2147-2155, 2012.

[JM04] P. Joshi, W. Maass. *Movement generation and control with generic neural microcircuits*. Lecture Notes in Computer Science, Biologically Inspired Approaches to Advanced Information Technology First International Workshop, BioADIT 2004, 3141, pp. 16–31, 2004.

[Jor86] M. I. Jordan. *Attractor Dynamics and Parallelism in a Connectionist Sequential Machine*. Proceedings of the Eighth Annual Conference of the Cognitive Science Society, pp. 531-546, Amherst, 1986.

[JSC00] F.A. Gers, J. Schmidhuber, F. Cummis. *Learning to forget: continual prediction with LSTM*. Neural Computation, 12(10), pp. 2451-2471, 2000.

[JSM97] J-S. R. Jang, C-T. Sun, E. Mizutani. *Neuro-Fuzzy and Soft Computing*. Englewood Cliffs, NJ: Prentice-Hall, 1997.

[KE95] J. Kennedy, R. Eberhart. *Particle swarm Intelligence*. Proceedings of IEEE International Conference on Neural Networks. IV. pp. 1942-1948, 1995.

[KGV83] S. Kirkpatrick, D. C. Gellat, M. P. Vecchi. *Optimization by Simulated Annealing*, Science, 220, pp. 671-680, 1983.

[Köh96] P. Köhn. *Genetic encoding strategies for neural networks*. Knowledge-Based Systems, 2, pp. 947-950, 1996.

[KSHB12] L. Khaouane, C. Si-Moussa, S. Hanini, O. Benkortbi. *Optimization of culture conditions for the production of Pleuromutilin from PleurotusMutilus using a hybrid method based on central composite design, neural network, and particle swarm optimization*. Biotechnology And Bioprocess Engineering, 17(5), pp. 1048-1054, 2012.

[LH88] K. J. Lang, G. E. Hinton. *The Development of the Time-Delay Neural Networks Architecture for Speech Recognition*. Technical Report CMU-CS, pp. 88-152, Carnegie-Mellon University, 1988.

[LJ09] M. Lukosevicius, H. Jaeger. *Reservoir computing approaches to recurrent*

neural network training. Computer Science Review, 3(3), pp. 127–149, 2009.

[LL11] N. F. Lima, T. B. Ludermir. *Frankenstein PSO Applied to Neural Network Weights and Architectures*. 2011 IEEE Congress on Evolutionary Computation, 2011, New Orleans. Proceedings of IEEE Congress on Evolutionary Computation. Los Alamitos: IEEE Computer Society, 1. pp. 1-6, 2011.

[LM07] R. Legenstein, W. Maass. *New Directions in Statistical Signal Processing: From Systems to Brain, chapter What makes a dynamical system computationally powerful?*, pp. 127–154. MIT Press, 2007.

[LYZ06] T. B. Ludermir, A. Yamazaki, C. Zanchettin. *An Optimization Methodology for Neural Network Weights and Architectures*. IEEE Transactions on Neural Networks, 17(5), pp. 1452-1459, 2006.

[ME04] R. Martí, A. El-Fallahi. *Multilayer neural networks: an experimental evaluation of online training methods*. Computers & Operations Research, 31, pp. 1491-1513, 2004.

[MLH03] D. Meyer, F. Leisch, K.Hornik. *The support vector machine under test. Neurocomputing*. 55(1), pp. 169–186, 2003.

[MNM02] W. Maass, T. Natschlager, H. Markram. *Real-time computing without stable states: A new framework for neural computation based on perturbations. Neural Computation*, 14(11), pp. 2531–2560, 2002.

[MNM03] W. Maass, T. Natschlager, H. Markram. *Advances in Neural Information* Processing Systems 15, NIPS 2002, volume 15, chapter A model for real-time computation in generic neural microcircuits., pp. 213–220. MIT Press, Cambridge, MA, 2003.

[MNM04] W. Maass, T. Natschlager, H. Markram. *Fading memory and kernel properties of generic cortical microcircuits models*. Journal of Physiology, 98(4-6), pp. 315–330, 2004.

[Moo98] C. Moore. *Dynamical recognizers: real-time language recognition by analog computers*. Theoretical Computer Science, 201, pp. 99-136, 1998.

[MP43] W. S. McCulloch, W. Pitts. *A Logical Calculus of the Ideas Immanent in Nervous Activity*. Bulletin of Mathematical Biophysics, 5, pp. 115–133, 1943.

[MSC+00] N. G. Metin, B. Sahiner, H. Chan, L. Hadjiiski, N. Petrick. *Optimal selection of neural network architecture for CAD using simulated annealing*. Proceedings of the 22th Annual EMBS International Conference, pp. 1325-

1330, 2000.

[OLSB05] M. Oubbati, P. Levi, M. Schanz, T. Buchheim. *Velocity control of an omnidirectional robocup player with recurrent networks*. In Robocup Symposium 2005, pp. 691–701, 2005.

[PC10] M. E. H. Pedersen; A. J.Chipperfield. *Simplifying particle swarm optimization*. Applied Soft Computing, 10, pp. 618–628, 2010.

[PD00] Y. P. Panos, J. W. Douglass. *Principles of optimal design: modeling and computation*. Cambridge University Press, 412p, 2000.

[Pea95] B. A. Pearlmutter. *Gradient calculation for dynamic recurrent neural networks: a survey*. IEEE Trans. On Neural Networks, 6(5), pp. 1212-1228, 1995.

[Pol91] J. B. Pollack. *The induction of dynamical recognizers*. Machine Learning, 7, pp. 227-252, 1991.

[PP07] S. Panda, N. P. Padhy. *Comparison of Particle Swarm Optimization and Genetic Algorithm for TCSC-based Controller Design*. International Journal of Computer Science & Engineering, 1(1), pp. 41, 2007.

[Sav98] J. E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley, Reading, MA, 1998.

[SDP+12] A. Smerieri, F.Duport, Y.Paquot, M.Haelterman, B.Schrauwen, M.Massar. *Towards Fully Analog Hardware Reservoir Computing For Speech Recognition*. International Conference of Numerical Analysis and Applied Mathematics (ICNAAM), 1479, pp.1892-1895, 2012.

[SDV+07] B. Schrauwen, J. Defour, D. Verstraeten, J. Van Campenhout. *The introduction of time-scales in reservoir computing, applied to isolated digits recognition.* LNCS, 4668(1), pp. 471–479, 2007.

[SE98] Y. Shi, R. C. Eberhart. *A modified particle swarm optimizer*. Proc. IEEE World Congr. Comput. Intell., pp. 69–73, 1998.

[SE99] Y. Shi, R. C. Eberhart. *Empirical study of particle swarm optimization*. Proc. IEEE Congr. Evol. Comput., pp. 1945–1950, 1999.

[SE01] Y.Shi, R. C. Eberhart. *Fuzzy adaptive particle swarm optimization.* Proc. IEEE Congr. Evol. Comput., pp. 101-106, 2001.

[SH12] M. Sheikhan, E. Hemmati. *PSO-Optimized Hopfield Neural Network-Based Multipath Routing for Mobile Ad-hoc Networks*. International Journal Of Computational Intelligence Systems, 5(3), pp. 568-581, 2012.

[SL11] D. N. G. Silva, T. B. Ludermir. *Otimização de Redes Neurais Usando GSO*

Cooperativos com Decaimento de Pesos. Simpósio Brasileiro de Inteligência Computacional, 2011, Fortaleza. Anais do Simpósio Brasileiro de Inteligência Computacional, pp. 1-8, 2011.

[SL12] A. Sergio, T. B. Ludermir. *PSO for reservoir computing optimization*. ICANN 2012, Part I, LNCS 7552, pp. 685-692, 2012.

[SMR12] B. Sohrabi,P. Mahmoudian, I.Raeesi. *A framework for improving e-commerce websites usability using a hybrid genetic algorithm and neural network system*. Neural Computing & Applications, 21(5), pp. 1017-1029, 2012.

[SP05] M. Salmen, P. G. Ploger. *Echo state networks used for motor control*. In International Conference on Robotics and Automation, ICRA 2005, pp. 1953–1958, 2005.

[SS94] H. Siegelmann, E. D. Sontag. *Analog computation via neural networks*. Theoretical Computer Science, 131, pp. 331-360, 1994.

[Ste04] J. J. Steil. *Backpropagation–decorrelation: Online recurrent learning with o(n) complexity*. In International Joint Conference on Neural Networks, IJCNN 2004, 2, pp. 843–848, 2004.

[Ste05] J. J. Steil. *Memory in backpropagation-decorrelation o(n) efficient online recurrent learning*. In 15th International Conference on Artificial Neural Networks, in: Lecture Notes in Computer Science, 3697, pp. 649–654, 2005.

[Ste06] J. J. Steil. *Online stability of backpropagation-decorrelation recurrent learninge*. Neurocomputing, 69, pp. 642–650, 2006.

[SVH12] B. Schrauwen, D. Verstraeten, and M. D Haene. *Reservoir Computing Toolbox Manual*. http://reslab.elis.ugent.be/rctoolbox, Dezembro 2012.

[SWGG07] J. Schmidhuber, D. Wierstra, M. E. Gagliolo, F. Gomez. *Training recurrent networks by evolino*. Neural Computation, 19(3), pp. 757–779, 2007.

[SWV+08] B. Schrauwen, M. Wardermann, D. Verstraeten, J. J. Steil, D. Stroobandt. *Improving reservoirs using intrinsic plasticity*. Neurocomputing, 71, pp. 1159–1171, 2008.

[Tre03] I. C. Trelea. *The Particle Swarm Optimization Algorithm: convergence analysis and parameter selection*. Information Processing Letters, 85, pp. 317–325, 2003.

[Van01] F. Van den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD thesis, University of Pretoria, Faculty of Natural and Agricultural Science, 2001.

[VDSD07] D. Verstraeten, B. Schrauwen, M. D'Haene, D. Stroobandt. *An experimental*

unification of reservoir computing methods. Neural Networks, 20(3), pp. 391–403, 2007.

[VFV+10] K. Vandoorne, M. Fiers, D. Verstraeten, B. Schrauwen, J. Dambre, P. Bienstman. Photonic Reservoir Computing: A New Approach to Optical Information Processing. 12th International Conference on Transparent Optical Networks (ICTON), Munich, German, 2010.

[VSSC05] D. Verstraeten, B. Schrauwen, D. Stroobandt, J. Van Campenhout. *Isolated word recognition with the liquid state machine: a case study*. Information Processing Letters, 95(6), pp. 521–528, 2005.

[Wai89] A. Waibel. *Modular Construction of Time-Delay Neural Networks for Speech Recognition*. Neural Computation, 1, pp. 39-46, 1989.

[Wai07] J. Wainer. *Métodos de pesquisa quantitativa e qualitativa para a ciência computação*. Atualização em Informática da Sociedade Brasileira de Computação, pp. 221–262, 2007.

[Wan90a] E. A. Wan*. Temporal Backpropagation for FIR Networks*. In Proc. IEEE Int. Joint Conf. Neural Networks, 1, pp. 575-580, 1990.

[Wan90b] E. A. Wan. *Temporal Backpropagation: an Efficient Algorithm for Finite Impulse Response Neural Networks*. In Proc. of the 1990 Connectionist Models Summer School, 1, pp. 131-140. 1990.

[Wer74] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[Wer90] P. Werbos. *Backpropagation Through Time: What It Does and To Do it*. In Proceedings of IEEE, 78, pp. 1550-1560, 1990.

[WF00] I. H Witten, E. Frank. *Data Mining, Practical Machine Learning Toolsand Techniques with Java Implementations*. Morgan Kaufmann Publishers, 2000.

[WHHSL89] A. Waibel, T. Hanazawa, J. G. Hinton, K.Shikano, K. Lang. *Phoneme Recognition Using Time-Delay Neural Networks*. IEEE AS-SP Magazine, 37, pp. 328-339, 1989.

[WLM12] WJ. Wu, SW. Lin, WK Moon. *Combining support vector machine with genetic algorithm to classify ultrasound breast tumor images*. Computerized Medical Imaging and Graphics, 36(8), pp. 627-633, 2012.

[WP89] R. J. Williams, J. Peng. *Reinforcment Learning Algorithm As Function Optimizers*. In International Joint Conference on Neural Networks, 2, pp. 89-95, Washington, 1989.

[WSVS08] F. Wyffels, B. Schrauwen, D. Verstraeten, D. Stroobandt. *Band-pass reservoir computing*. In International Joint Conference on Neural Networks 2008, pp. 3203–3208, 2008.

[WZ89] R. J. Williams, D. Zipser. *A Learning Algorithm for Continually Running Fully Recurrent Neural Networks*. Neural Computation, 1, pp. 270-280, 1989.

[Zan08] C. Zanchettin. Otimização Global em Redes Neurais Artificiais. Tese de doutorado. Universidade Federal de Pernambuco. Cin. Ciência da Computação, 2008.

[ZCL07] J. Zhang, H. S.-H. Chung, W.-L. Lo. *Clustering-based adaptive crossover and mutation probabilities for genetic algorithms*. IEEE Trans. Evol. Comput., 11(3), pp. 326–335, 2007.

[ZXZC07] Z. H. Zhan, J. Xiao, J. Zhang, W. N. Chen. *Adaptive control of ac-celeration coefficients for particle swarm optimization based on clustering analysis*. Proc. IEEE Congr. Evol. Comput., pp. 3276–3282, 2007.

[ZZLC09] Z-H. Zhan, J. Zhang, Y. Li, H. Chung, H. *Adaptive Particle Swarm Optimization*. IEEE Transactions, Man and Cybernetics, 39, pp. 1362-1381, 2009.