

UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação
ISSN 0103-2569

**ESTUDO SOBRE ABORDAGENS DE EXTRAÇÃO, CLASSIFICAÇÃO E
PREDIÇÃO DE COMPORTAMENTO DE PROCESSOS**

**EVGUENI DODONOV
RODRIGO FERNANDES DE MELLO**

Nº 322

RELATÓRIOS TÉCNICOS



**São Carlos – SP
Mai./2008**

SYSNO	1666658
DATA	/ /
ICMC - SBAB	

Estudo sobre abordagens de extração, classificação e predição de comportamento de processos

Evgueni Dodonov

Rodrigo Fernandes de Mello

12 de maio de 2008

"The only way to see the future is to create it."
George C. Orwell

"Those who control the past, control the future"
George C. Orwell

Resumo

A área de predição do comportamento de processos têm despertado atenção nas últimas décadas, resultando em novas abordagens, técnicas e aplicações.

Este relatório técnico tem como o objetivo introduzir e descrever diversas técnicas que podem ser utilizadas para determinar e avaliar o comportamento de processos e prever suas operações futuras.

Enquanto este material é voltado à predição do comportamento de processos em sistemas distribuídos, as técnicas estudadas podem ser aplicadas em diversas áreas computacionais.

Sumário

1	Introdução	4
2	Estado da arte e trabalhos relacionados	6
2.1	Considerações iniciais	6
2.2	Avaliação do comportamento de processos distribuídos	6
2.3	Avaliações e comparações entre as técnicas	8
2.4	Aplicações	9
3	Extração do comportamento de aplicações	11
3.1	Considerações iniciais	11
3.2	Organização e representação do comportamento de processos	11
3.2.1	Técnicas de extração de comportamento	13
3.2.2	Representação do comportamento de processos	13
3.2.3	Ferramentas para extração do comportamento	14
3.2.4	Ferramentas para avaliação de aplicações <i>MPI</i>	15
3.3	Aplicações	16
3.4	Considerações finais	17
4	Classificação de padrões	18
4.1	Considerações iniciais	18
4.2	Técnicas estocásticas	18
4.2.1	Modelo de <i>k-médias</i>	19
4.2.2	Clusterização <i>fuzzy</i> de <i>c-médias</i>	19
4.2.3	Algoritmo de clusterização <i>QT</i>	21
4.2.4	Classificação linear: <i>SVM (Support Vector Machines)</i>	22
4.3	Técnicas baseadas em redes neurais	23
4.3.1	Mapas auto-organizáveis – <i>SOM</i>	26
4.3.2	Redes neurais auto-expansíveis	28
4.3.3	<i>Cascade-Correlation Learning Architecture</i>	29
4.3.4	<i>Growing Cell Structure</i>	29
4.3.5	<i>Growing Neural Gas</i>	30
4.3.6	<i>Growing Self-Organizing Maps</i>	31
4.3.7	<i>Restricted Coulomb Energy</i>	31
4.3.8	<i>Contextual Layered Associative Memory</i>	31
4.3.9	<i>Growing When Required</i>	31
4.3.10	<i>Adaptive Resonance Theory</i>	34

4.3.11	Rede neural <i>SONDE</i>	39
4.3.12	<i>Radial Basis Function</i>	40
4.3.13	Redes neurais separadoras	40
4.4	Considerações finais	43
5	Predição do comportamento de processos	44
5.1	Considerações iniciais	44
5.2	Técnicas baseadas no estado atual	44
5.2.1	Cadeias de Markov	45
5.2.2	Modelo Markov escondido	46
5.2.3	Modelo <i>HMM</i> Hierárquico	47
5.2.4	Modelo <i>HMM</i> em níveis	48
5.3	Técnicas auto-regressivas	48
5.3.1	Auto-regressão linear	49
5.3.2	Modelos não-lineares	50
5.3.3	Auto-correlação	51
5.4	Filtro de Kalman	51
5.4.1	Modelo de Kalman suavizado: <i>Kalman Smoother</i>	53
5.4.2	Filtro de Kalman estendido	55
5.5	Redes neurais recorrentes	56
5.5.1	Rede <i>LSTM</i>	57
5.5.2	Rede neural de <i>Hopfield</i>	61
5.5.3	Técnica de <i>Simulated annealing</i>	64
5.5.4	Máquinas de Boltzmann	64
5.6	Redes Associativas Competitivas	65
5.6.1	Modelo <i>SOM</i> temporal	66
5.6.2	Modelo <i>SOM</i> auto-regressivo	66
5.6.3	Rede neural <i>VQTAM</i>	67
5.7	Predição por meio de atrasos no tempo: <i>TDNN</i>	68
5.8	Aprendizado supervisionado: <i>Reservoir computing</i>	69
5.9	Aprendizado controlado: <i>SRNN</i>	71
5.10	Aprendizado Bayesiano	72
5.11	Teoria de informação e entropia	75
5.12	Computação bio-inspirada: algoritmos genéticos	77
5.13	Considerações finais	78
6	Conclusões	79

Lista de Figuras

3.1	Organização do comportamento dos processos	12
4.1	Classificação de acordo com o modelo de k -médias	20
4.2	<i>SVM</i> – separação entre as classes	22
4.3	Componentes de uma rede neural	24
4.4	Funções de ativação de neurônios	25
4.5	Topologia de rede neural	26
4.6	SOM: agrupamento de dados	27
4.7	ART: classificação de dados	35
4.8	Arquitetura da rede ART 2A	37
4.9	Rede neural <i>RBF</i>	41
4.10	Arquitetura da rede <i>RBF</i>	41
4.11	ICA: decomposição de sinais	42
5.1	Exemplo de cadeia de Markov	46
5.2	Exemplo de <i>HMM</i>	47
5.3	Exemplo de <i>HHMM</i>	48
5.4	Exemplo de <i>LHMM</i>	49
5.5	Exemplo de filtro de Kalman	52
5.6	Exemplo de uma rede neural recorrente	56
5.7	Rede neural de Elman	57
5.8	Célula de memória de <i>LSTM</i>	59
5.9	Exemplo de <i>LSTM</i>	61
5.10	Rede <i>TDNN</i>	69
5.11	Arquitetura de <i>Reservoir Computing</i>	70
5.12	Spiral Recurrent Neural Network	72
5.13	Exemplo de rede Bayesiana	74
5.14	Exemplo de árvore de dependências da 2a ordem	74
5.15	Cadeia de Markov para cálculo de entropia	76

Capítulo 1

Introdução

A disponibilidade de microprocessadores de baixo custo e a evolução das redes de computadores motivou o desenvolvimento de sistemas distribuídos, possibilitando a realização de um mesmo trabalho computacional sobre elementos de processamento (*EPs*), interconectados por meio de uma rede computacional [35]. Diversos conceitos de sistemas distribuídos têm sido adotados desde a década de 1980 para a resolução de tais problemas, uma vez que este tipo de sistemas pode oferecer alto desempenho a um custo menor, quando comparado com super-computadores ou máquinas paralelas. Com isso, novas técnicas voltadas para a execução eficiente de aplicações distribuídas foram concebidas, tais como mecanismos de distribuição eficiente de tarefas pela rede (balanceamento de carga e migração de processos) [18, 50, 15], acesso eficiente a dados distribuídos (protocolos de baixa latência e dispositivos de hardware específicos) [68, 101, 124] e mecanismos de otimização de tráfego (pré-busca e mecanismos de *caching*) [12, 68].

Entretanto, a necessidade da adaptação de aplicações paralelas à arquiteturas, bibliotecas ou ambientes específicos têm sido um dos maiores problemas para a utilização de aplicações legadas em tais ambientes de forma eficiente. Para isso, a tecnologia conhecida como imagem única do sistema (*SSI – Single System Image*) foi introduzida, que visa representar um sistema distribuído como sendo um local, deixando a distribuição de dados e execução transparente para as aplicações do usuário [15].

Com isso, a possibilidade da utilização de ambientes distribuídos para aplicações convencionais tornou-se mais viável, uma vez que o sistema, de forma transparente, pode distribuir e realizar as tarefas computacionais dos usuários automaticamente. Ambientes de balanceamento de carga e migração de processos, tais como *PBS* [18], *Cosmic* [50], *TUI* [208] e *Mosix* [15], e sistemas de memória compartilhada distribuída (*DSM - Distributed Shared Memory*) [22, 134, 8, 88] possibilitam a execução e comunicação transparente de aplicações sobre um ambiente distribuído, geralmente composto por máquinas homogêneas e pré-configuradas para a execução das aplicações específicas dos usuários.

Diversos fatores influenciam na utilização eficiente desses ambientes, tais como as políticas de escalonamento e balanceamento de carga [169, 93, 199], protocolos de baixa latência [64, 73] e mecanismos de entrada e saída [104, 68], mencionados anteriormente. A dependência desses fatores das características específicas de cada aplicação, como utilização de recursos computacionais, memória, acessos a disco e rede, motivou estudos para distribuir e prever o comportamento de aplicações [199, 198, 110, 32, 179, 203]. Esses estudos demonstraram que a predição do comportamento futuro das aplicações tem um grande potencial para a otimiza-

ção do desempenho do sistema, tendo aplicações em todas as áreas de computação distribuída mencionados acima [72, 71].

Diversas técnicas foram propostas para prever o comportamento futuro de aplicações, variando entre abordagens estatísticas, modelos matemáticos, redes neurais e computação bio-inspirada.

Este trabalho tem como o objetivo a descrição de principais conceitos e técnicas que podem ser utilizadas para extrair, classificar e prever o comportamento de aplicações, e é organizado da seguinte forma: capítulo 2 apresenta alguns dos trabalhos relacionados à área de predição do comportamento de processos e suas aplicações em ambientes distribuídos. As principais abordagens de extração do comportamento de processos são apresentadas no capítulo 3. Técnicas de classificação e predição do comportamento são apresentadas nos capítulos 4 e 5. Finalmente, capítulo 6 conclui este trabalho.

Capítulo 2

Estado da arte e trabalhos relacionados

2.1 Considerações iniciais

A predição do comportamento de aplicações tem o potencial de aumentar significativamente o desempenho de diversas áreas de computação [72, 71, 101], podendo auxiliar no balanceamento de carga, escalonamento de processos e leitura antecipada de dados, resultando em elaboração de diversos trabalhos, abordando as mais variáveis táticas de predição.

Este capítulo visa apresentar alguns dos trabalhos relacionados à classificação e predição do comportamento de processos em ambientes distribuídos, ilustrando os diversos caminhos e decisões tomadas pelos seus autores, demonstrando as suas aplicações, avaliações e comparações com as abordagens atuais.

O capítulo é organizado em seguintes seções. A seção 2.2 aborda os trabalhos que avaliam o comportamento de processos distribuídos para realização de tarefas computacionais. Trabalhos referentes a avaliações e comparações entre diversas técnicas de classificação e predição são apresentados na seção 2.3. E, finalmente, algumas aplicações práticas de técnicas de predição do comportamento são apresentados na seção 2.4.

2.2 Avaliação do comportamento de processos distribuídos

Diversos trabalhos tem sido desenvolvidos com o objetivo de determinar e prever futuras ações de processos com base na avaliação estatística do seu comportamento.

Estudos sobre as características de acesso ao sistema de arquivos em sistemas multiprocessados são apresentados em [150, 181]. Nesses trabalhos, são avaliados as características de acessos a dados por todos os processos no sistema durante um período de duas semanas. Baseando-se nos resultados obtidos, foi demonstrado que a maioria de acessos ocorre de maneira repetitiva, possibilitando prever e antecipar acessos futuros observando a frequência e taxa de acessos a dados. Com isso, o trabalho demonstra que a utilização de um mecanismo de predição conjunto com sistema de *caching* pode otimizar significativamente o desempenho do sistema [150].

Diversos trabalhos relacionados à predição de acessos são de autoria de Cortes *et al* [60, 58, 59, 56, 55, 57]. Os trabalhos utilizam um sistema de arquivos paralelo *PAFS* [60, 57], aplicando diversas técnicas de predição de acessos em técnicas de *cache* e *prefetching*. No sistema, os mecanismos de *cache* são organizados em dois níveis – uma série de *caches* locais

para cada elemento de processamento do sistema, e um *cache global* que consiste de um espaço de memória único e compartilhado. A consistência de *caches* é mantida utilizando *tokens* repassados entre *caches* distintos.

Cortes *et al.* estendem a discussão do PAFS em [58], avaliando diferentes tipos de acesso ao *cache* – *local cache hit*, *remote cache hit* e *global cache hit*. A utilização de servidores dedicados para *cache* (*cache servers*) é apresentada e discutida no trabalho. Visando manter a consistência entre diversos *caches* e diminuir a comunicação entre servidores, é introduzido um mecanismo de *hashing* para determinar de maneira única a localização de um bloco de dados em diversos servidores de *cache*. No sistema, a predição de acessos é feita por meio de técnica de cadeias de Markov, avaliando a taxa e frequência de acessos a diversos *caches* para arquivos independentes.

Uma outra avaliação do modelo estocástico de Markov é apresentada em [204]. Diferentemente de outros trabalhos, o trabalho visa prever a ordem de acessos a arquivos independentes, e não aos blocos de dados dentro de um mesmo arquivo. Os autores introduzem e apresentam quatro abordagens para a determinação do padrão de acesso à arquivos – *First Successor*, *Last Successor*, *Stable Successor* e *Best-k-out-of-m*. O objetivo das técnicas estudadas é o agrupamento de arquivos acessados concorrentemente. Como resultado das avaliações, os autores afirmam que é possível antecipar até 80% de todos os acessos a arquivos.

Um estudo de predição de acessos em bases de dados distribuídas é realizado em [232], apresentando um modelo analítico para a predição de acessos a *buffers* por meio de avaliações estocásticas. Como conclusões do trabalho, os autores afirmam que a predição de acesso a *buffers* pode aumentar significativamente a quantidade de transações por segundo realizados.

Entre os trabalhos relacionados à predição de comportamento de processos por meio da avaliação do histórico de operações passadas é possível destacar os trabalhos de Kots *et al.*, que tratam da avaliação e predição do comportamento de aplicações em sistemas distribuídos, com o objetivo de extrair diversos padrões de acesso [147, 146, 149, 151, 145, 152, 150, 181].

Diversas técnicas de reconhecimento de padrões sequenciais de acesso são apresentados em [146, 147], apresentando padrões de acesso tais como *one-block look-ahead*, *infinite-block look-ahead* e *portion recognition*. O algoritmo *predictor* é introduzido pelo trabalho com o objetivo de detectar automaticamente do tipo de acesso atual da aplicação.

Tratando-se da predição e antecipação de acessos a dados, é de grande importância o trabalho de Cao *et al.* [38], que discute as possíveis implementações dos mecanismos de predição, avaliando os algoritmos de *prefetching* *agressivo* e *passivo*, apresentando as vantagens e desvantagens de cada abordagem. Nesse trabalho, as principais abordagens para a implementação de um mecanismo de *prefetching* são discutidas, definindo as regras básicas para um mecanismo de leitura antecipada eficiente – *Optimal Prefetching*, *Optimal Replacement*, *Do No Harm* e *First Opportunity* [38, 68].

Tópicos de grande relevância para a área de antecipação de acessos a dados são apresentados em [227, 138, 139]. Os trabalhos introduzem o termo *prefetching*, definindo-o como *mecanismo para a redução do tempo ocioso por meio de predição e antecipação de acessos*, e avaliam diversos algoritmos de predição e leitura antecipada de dados, tais como *forestall*, *fixed horizon*, *aggressive* e *reverse aggressive*. De acordo com os experimentos realizados, a predição e antecipação de acessos aos dados possibilitam aumentar significativamente o desempenho do sistema, diminuindo a latência dos acessos. Os trabalhos introduzem um sistema de predição e antecipação de acessos para memória global PGMS (*Prefetching Global Memory System*) – um sistema que integra os *caches locais* de cada servidor em um *cache global*, visível

para todos os servidores, facilitando a troca de informações entre os *EPs*.

De acordo com Soloviev *et al.* [211], é possível utilizar a predição de acessos para melhorar o desempenho de discos rígidos, analisando e re-organizando os acessos paralelos de acordo com a estrutura física do disco. Abordagem similar é aplicada em sistemas de arquivos paralelos por Aranachalam *et al.* [10].

Barve *et al.* [19] avalia dois mecanismos de predição de acessos por meio de *prefetching* – *NOM* e *GREED*. O algoritmo *NOM* avalia constantemente o *look-ahead* global do sistema distribuído enquanto o *GREED* utiliza o *look-ahead* local para determinar os próximos blocos da sequência a serem lidos. De acordo com a previsão do algoritmo, a sequência de acessos a blocos é re-ordenada, diminuindo o número de requisições e melhorando a latência de acessos aos dados.

Um outro algoritmo de otimização de acessos aos dados é apresentado em [130], que funciona associando prioridades diferentes para todas as requisições de leitura de dados previstas, resultando em melhor desempenho das operações de entrada e saída por meio de reorganização das requisições.

A aplicação de mecanismos de predição em sistemas de memória distribuída é discutida em [24, 25]. Os trabalhos apresentam os algoritmos de predição *B+* e *Adaptive++*, que utilizam o histórico de acessos à memória e as invalidações dos blocos em *cache* para determinar a lista de blocos a serem requisitados no futuro.

Considerações sobre otimizações de acesso aos dados por meio de predições de acessos sequenciais e contínuos são apresentados em [212, 158, 68], sendo aplicados a um sistema de arquivos paralelo e distribuído *NPFS* [104]. Otimizações para a latência de operações de entrada e saída são propostas em [212], visando adaptar de forma dinâmica os *timeouts* de mensagens aos parâmetros do meio de comunicação utilizado, e são avaliadas por um sistema de vídeo sob demanda distribuído [158]. Estudos sobre predição de acessos em sistemas de arquivos paralelos são realizados em [68], apresentando um sistema de *cache* e *prefetching* adaptativo e auto-configurável. No trabalho, dois novos algoritmos de *prefetching* – *limited aggressive* e *prefetch-on-empty* – são propostos e avaliados junto com algoritmos *aggressive* e *passive* tradicionais. O trabalho também introduz o algoritmo de predição de acessos sequenciais *CPS*.

2.3 Avaliações e comparações entre as técnicas

Diversos trabalhos foram desenvolvidos para detectar padrões de acessos de forma automática, tais como os de Sakr *et al.* [193], que avaliam e comparam três métodos de detecção do padrão de acesso – um baseado em abordagem estocástica de Cadeias de Markov; um baseado em predição linear e um método que utiliza redes neurais (*TDNN*). O trabalho comparou a eficiência dos mecanismos para a predição de acessos a curto prazo (*one-step-ahead prediction*).

O trabalho conclui que diferentes abordagens são mais apropriadas para diversos tipos de aplicações. Por exemplo, enquanto em um caso a predição baseada em cadeias de Markov e predição linear conseguiram eliminar 95% de predições erradas e *TDNN* conseguiu eliminar 71% de falhas, em outro caso o preditor linear conseguiu remover somente 1% de falhas, o baseado em cadeias de Markov removeu 6% de falhas e o baseado em *TDNN* conseguiu remover 30% de falhas.

Como conclusão final, os autores afirmam que, enquanto todas as técnicas apresentam uma

eficiência compatível para padrões de acesso simples, com o aumento do grau de complexidade de acessos os modelos simples passam a se tornar ineficientes, e o modelo baseado em redes neurais (*TDNN*) oferece a melhor eficiência.

A predição de comportamento de processos em ambiente multi-processado é avaliada em [193]. No trabalho, rede *TDNN* é utilizada para prever seqüências de acessos à memória em processadores independentes. Três aplicações diferentes são utilizadas como estudo de caso – *temperature propagation/2D relaxation*, multiplicação de matrizes e transformada de Fourier. De acordo com os autores, a rede neural *TDNN* possibilitou diminuir em até 3.28 vezes o número de acessos à memória desnecessários, antecipando-os.

Pesquisas realizadas por Kroeger *et al.* [154] demonstraram que a maioria de acessos a dados em sistemas distribuídos ocorre de forma seqüencial, sendo que a predição de acessos futuros baseando-se no histórico de acessos seqüenciais passados pode ser feita com a precisão de aproximadamente 70 – 80%. O trabalho apresenta três mecanismos de predição de acessos a arquivos independentes, comparando a sua eficiência.

Um estudo sobre predição de acessos em sistemas distribuídos é apresentado em [36], focando em predição de acessos a dados por aplicações baseadas em *MPI* e apresentando um modelo para a predição de acessos baseado em padrões de funcionamento de aplicações distribuídas. O modelo proposto possibilita prever 80 – 90% de todos os acessos, otimizando a latência de acessos à memória.

Finalmente, é de grande relevância as competições entre diversas técnicas de predição aplicadas à séries temporais realizadas em [231, 77, 45, 218], avaliando diversas técnicas de predição com o objetivo de determinar a mais adequada para as predições de curto e longo prazo.

2.4 Aplicações

A evolução constante e rápida de arquiteturas de processadores torna ineficientes os métodos convencionais de acesso aos dados, limitados pelas configurações do meio de comunicação utilizado. Visando solucionar esse problema, diversas estratégias têm sido propostas, abordadas a seguir.

Mecanismos de *prefetching*, também conhecidos como *leitura antecipada* ou *redução de tempo ocioso do sistema* [227], procuram antecipar o acesso a dados, pré-carregando-os num espaço de armazenamento, tal como a memória principal ou *cache*. A utilização conjunta desses mecanismos possibilita diminuir a latência de acesso a dados, antecipando requisições ao meio de armazenamento. Entretanto, a eficiência dos mecanismos de *prefetching* é limitada pela necessidade de determinar de forma correta o comportamento, ou padrões de acesso, das aplicações.

Uma arquitetura de memória compartilhada distribuída voltada para jogos multi-usuário é apresentada em [195], utilizando o sistema operacional *Plurix OS*, baseado na plataforma Java com sistema de endereçamento de memória global. O mecanismo de memória compartilhada utilizado é baseado no conceito de transações atômicas para manter a consistência de dados.

Nesse sistema, a memória compartilhada é utilizada para armazenar as informações sobre o ambiente distribuído de maneira consistente. Cada participante de ambiente requisita informações atualizadas sobre o ambiente da memória compartilhada periodicamente, atualizando os seus dados locais, e variando a periodicidade destas requisições de acordo com a capacidade de processamento de cada *EP*. Dessa forma, pode-se diminuir a taxa de atualizações de dados nos

computadores com menor poder de processamento, ao mesmo tempo mantendo uma visão consistente do ambiente distribuído. Por exemplo, no caso de um objeto se movendo no ambiente distribuído, computadores mais potentes podem visualizar o movimento completo do mesmo e os computadores com menor poder de processamento podem visualizar somente alguns dos trechos de movimento. A predição de tais eventos é essencial para a utilização eficiente do sistema.

Um mecanismo de *memory ushering*, que visa a utilização da memória de processadores distintos através de migração de processos ou *swapping*, é introduzido em [14]. A principal diferença com os modelos tradicionais de *DSM* consiste na utilização não-uniforme da memória, sendo que o algoritmo de distribuição de acessos à memória visa colocar um número máximo de processos ativos no mesmo *cluster*, evitando *thrashing* ou *swapping*. Um algoritmo de balanceamento de carga convencional é utilizando enquanto a memória livre de um processador não ultrapassa um pré-determinado valor. Nesse caso, um algoritmo diferente é ativado, migrando o processo para um elemento de processamento com memória disponível. O trabalho demonstrou que a predição de acessos às páginas de memória é essencial para o desempenho do sistema.

Martin *et al.* [168] adotam o conceito de conjuntos de destino para otimizar a consistência de dados. Um conjunto de destino é definido como um conjunto de elementos de processamento que recebem uma mesma requisição para garantir consistência de memória. O trabalho propõe a adoção de duas técnicas: a primeira baseada em protocolos *snoopy* que visa enviar as requisições de consistência de memória para um conjunto máximo de *EPs*, permitindo diminuir a latência de *sharing misses* ao custo de uma largura de banda maior. A segunda é baseada na utilização de um serviço de diretório centralizado que envia requisições para um conjunto mínimo de *EPs* para diminuir a largura de banda necessária. A predição de acessos à memória compartilhada possibilita diminuir significativamente a latência e a largura de banda necessários.

Finalmente, estudos sobre aplicações de técnicas de predição de acessos a dados distribuídos em mecanismos de leitura antecipada e balanceamento de carga são apresentados em [72, 71], propondo um modelo para predição do comportamento de processos por meio de classificação e predição de seus acessos. O trabalho separa o processo de predição de acessos em três estágios: o de extração do comportamento de acessos, o de classificação e agrupamento de acessos similares e o de predição. Diversas técnicas podem ser utilizadas em cada um dos estágios, variando entre abordagens estatísticas e baseadas em redes neurais. Em [72], redes neurais *ART2A* e *TDNN* são avaliadas, avaliando a sua eficiência sobre o *NPB (NAS Parallel Benchmark)*.

Capítulo 3

Extração do comportamento de aplicações

3.1 Considerações iniciais

Diversas pesquisas demonstraram que a previsão de comportamento de um processo tem o potencial de melhorar significativamente a eficiência de execução de processos em ambientes distribuído [72, 170, 71]. A determinação correta do comportamento de um processo é, portanto, essencial para a predição das suas ações futuras, podendo ser utilizada por diversas técnicas de otimização do desempenho e eficiência do sistema.

Para isso, é necessário avaliar o comportamento do processo, detectando seus padrões de execução e acesso a dados. Tratando-se de acessos sequenciais, o padrão de acesso pode ser determinado de maneira simples [148]. Entretanto, a determinação de um padrão de acesso não-sequencial necessita de técnicas ou ferramentas especiais [72, 71].

Este capítulo apresenta as técnicas que podem ser aplicadas para a extração e representação desse comportamento.

3.2 Organização e representação do comportamento de processos

O comportamento de processos pode ser representado de acordo com diversos parâmetros que representam as suas características fundamentais. De forma geral, o conhecimento sobre o comportamento de processos pode ser organizado em níveis global e isolado, conforme demonstrado na figura 3.1.

O nível global tem como o objetivo o agrupamento de todas as informações sobre as aplicações em execução no sistema, tendo como as características comuns o seu tempo de execução e sua relação com a carga total do sistema. Estes parâmetros podem ser utilizados para mapear o tempo de execução de uma aplicação específica em função de quantidade de recursos consumidos por ela, quantificando tais recursos de forma previsível.

Essas características podem ser organizadas em classes, por exemplo, agrupando-as pelo nome do usuário ou da aplicação. Esta abordagem é amplamente utilizada em ambientes de grades, sendo a base de diversos modelos de *grid economy* [1].

O nível isolado, por sua vez, representa o comportamento da aplicação em função das tarefas específicas exercidas por ela, tais como a interação com usuários e utilização de recursos computacionais.

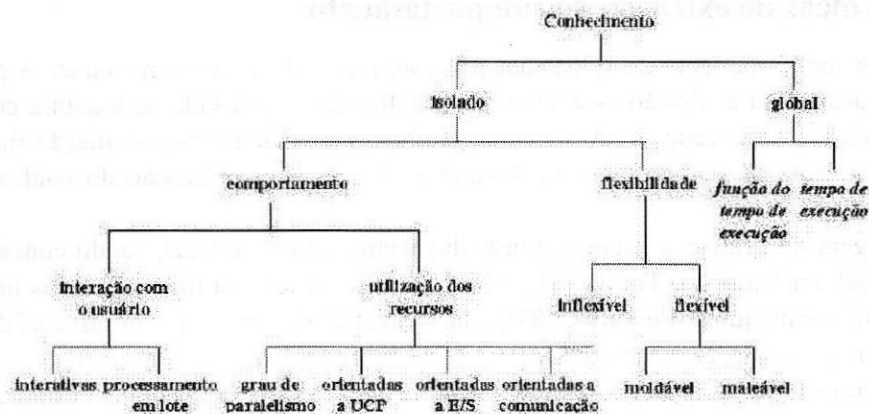


Figura 3.1: Organização do comportamento dos processos

O comportamento de aplicações em relação à interação com usuário pode ser classificado como interativo, para aplicações que precisam de interação com o usuário; e não-iterativas, também conhecido como processamento em lote, para as que não necessitam de interação explícita.

O padrão de utilização de recursos permite classificar as aplicações em seguintes classes: as orientadas à processamento (*CPU-Bound*), representando as aplicações voltadas para cálculos e processamento de dados; as orientadas a Entrada e Saída (*IO-Bound*), agrupando aplicações com predominância de transferências e acessos a dados; e as voltadas para comunicação (*Communication Intensive*), que abrange processos caracterizados por troca excessiva de mensagens com outros elementos de processamento.

Além disso, é possível classificar as aplicações de acordo com o grau de paralelismo, determinando a capacidade da utilização de mais de um elemento de processamento em paralelo. Neste contexto, o grau de paralelismo representa o número de *EPs* que maximiza a eficiência da execução da aplicação. Da mesma forma, o grau de paralelismo mínimo determina a quantidade mínima de *EPs* no sistema para a execução da aplicação; e o paralelismo máximo representa o número máximo de elementos de processamento que podem ser utilizados pela aplicação de forma eficiente.

A classificação de aplicações de acordo com a flexibilidade determina a capacidade de adaptação a ambientes variáveis, representando a sua eficiência em relação às decisões tomadas pelo escalonador do sistema na atribuição de recursos do sistema [85]. Uma aplicação inflexível, neste contexto, sempre necessita de uma configuração de ambiente pré-determinada, sendo que sua execução em um sistema diferente é inviável ou difícil. Aplicações flexíveis, por sua vez, podem ser classificadas em moldáveis e maleáveis.

As aplicações moldáveis são caracterizadas pela possível adaptação ao ambiente somente no início da execução. As maleáveis, entretanto, permitem uma flexibilidade ainda maior, adaptando-se à configuração do ambiente durante a execução. As aplicações flexíveis geralmente são construídas utilizando o modelo de paralelismo de dados.

3.2.1 Técnicas de extração de comportamento

A extração de comportamento de aplicações pode ser realizada de duas maneiras. A primeira abordagem consiste na avaliação estática, onde a aplicação é analisada de maneira empírica, sem a necessidade de execução. A segunda, também conhecida como avaliação dinâmica, consiste na extração de comportamento durante execução, sem utilização do conhecimento prévio.

A abordagem de avaliação estática é uma das técnicas mais antigas, sendo concebida na década de 1930 por Church e Turing [87]. Originalmente, a técnica foi aplicada às máquinas de estados finitos e máquinas de Turing [87], com o objetivo de determinar possíveis falhas que podem levar a um estado de empasse, finalizando a execução de maneira imprevista.

Com a evolução de sistemas computacionais, novas técnicas de avaliação estática foram propostas. Atualmente, entre tais técnicas pode-se citar o método de verificação de modelo [177, 76, 100, 111], que consiste na redução da aplicação para uma representação formal através de um autômato; e o método de interpretação abstrata [51, 115, 164, 228], que consiste na avaliação do sistema modelando-o na forma de uma máquina de estados abstrata, representando cada declaração por um estado.

A técnica de avaliação dinâmica, por sua vez, permite investigar o comportamento da aplicação durante a sua execução, geralmente com o auxílio de ferramentas de monitoramento ou depuração. É possível dividir as técnicas de avaliação dinâmica nas de monitoramento contínuo e baseadas em eventos [128].

Técnicas de monitoramento contínuo consistem na avaliação periódica do estado de aplicações em execução, permitindo extrair as variações do comportamento. Essa técnica geralmente é mais simples de ser utilizada, uma vez que não necessita de alterações em aplicações. Contudo, sua desvantagem está relacionada ao fato de que o monitoramento ocorre continuamente, introduzindo um *overhead* na execução. Outra desvantagem dessa abordagem está relacionada ao fato de não possibilitar a extração do comportamento de aplicações em estados de execução específicos.

A técnica baseada em eventos avalia o estado de execução de aplicações em estados pré-definidos, permitindo uma melhor extração de comportamento, uma vez que apenas ações em etapas relevantes são capturadas para posterior análise. Geralmente, esta técnica apresenta maior complexidade para adoção, necessitando de conhecimento prévio sobre o funcionamento das aplicações avaliadas. Além disso, a necessidade de adaptar aplicações para o processo de interceptação de eventos (através de instrumentação ou interceptação de chamadas) torna essa técnica mais complicada para uso.

Como vantagem dessa abordagem pode-se citar a determinação do comportamento de aplicações em períodos de execução específicos (por exemplo, nos pontos de sincronização entre processos distribuídos). Além disso, o *overhead* introduzido na execução é menor, uma vez que a avaliação do comportamento não é contínua.

3.2.2 Representação do comportamento de processos

A avaliação do comportamento histórico de uma aplicação permite prever suas próximas operações. Para isso deve-se analisar as mudanças de comportamento e frequência dessas alterações.

Geralmente, o comportamento de processos ao longo do tempo é representado por meio

de *séries temporais*, um conjunto de observações feitas sequencialmente ao longo do tempo. Enquanto em modelos de regressão linear a ordem das observações é irrelevante para a análise [136], tal ordem é fundamental para análise de séries temporais, uma vez que as observações vizinhas geralmente são inter-dependentes. Desse modo, uma série temporal pode ser utilizada para analisar e modelar o comportamento de um processo, avaliando a dependência entre observações subseqüentes e possibilitando a previsão de acontecimentos futuros.

Duas abordagens podem ser empregadas para estudar uma série temporal. A primeira abordagem define a análise da série como sendo um método para entender o comportamento do processo que a gerou. A outra abordagem visa realizar previsões a partir da série, construindo modelos matemáticos a partir de quais é possível estimar e prever acontecimentos futuros.

Por exemplo, o comportamento de um processo pode ser representado por meio de uma série de transições entre estados **em execução** e **ocioso** ao longo de tempo. Representado o estado **em execução** como 1, e estado **ocioso** como 0, a série temporal que representaria o comportamento de tal processo durante $n = 10$ intervalos de amostragem pode ser representada como $X = [X_1, X_2, \dots, X_n] = [0, 1, 0, 1, \dots, 1]$.

Dessa forma, uma série temporal representa o conjunto sequencial de eventos, podendo ser reduzida para uma equação obtida por meio de uma regressão linear ou não-linear.

3.2.3 Ferramentas para extração do comportamento

Umas das tarefas mais comuns na avaliação do comportamento de aplicações é a análise do seu código fonte. As abordagens mais comuns consistem na avaliação manual de código, utilizando ferramentas como *grep*, *find* e *sed* [69].

A utilização de aplicações para indexação de código fonte, como *LXR*, *Doxygen* e *JavaDoc*, possibilita visualização mais eficiente do comportamento, possibilitando observar as interações entre classes, funções e trocas de mensagens. Enquanto algumas dessas ferramentas necessitam de uma instrumentação específica de código fonte por meio de comentários, como *JavaDoc* e *Doxygen*, ferramentas como *LXR* realizam indexação automaticamente.

A avaliação dos problemas presentes nas aplicações pode ser feita de maneira automática, utilizando ferramentas como *Lint*, *Blast* e *Valgrind*, que permitem identificar falhas mais comuns em aplicações, tais como uso incorreto de memória, *casting* incorreto de parâmetros, etc. Uma outra abordagem consiste em adaptação automática do código fonte de aplicações, determinando, analisando e modificando seus pontos críticos. Um modelo semi-automático de transposição e avaliação de código fonte foi proposto por Quadrado [187], com o objetivo de auxiliar na avaliação e transformação de aplicações sequenciais em paralelas.

Para avaliar o comportamento de aplicações de maneira dinâmica, várias técnicas podem ser empregadas. Dentre elas está a de *tracing*, baseada em visualização da seqüência das chamadas do sistema feitas pela aplicação. Essa técnica é utilizada nas ferramentas *strace* e *GridBox* [74].

Nessa abordagem é utilizada uma chamada do sistema específica – *ptrace* [69] – que indica ao núcleo do sistema operacional que um processo está em depuração. Dessa maneira, o sistema passa a controlar a execução do processo, rastreando chamadas de sistema, e permitindo executar operações arbitrárias a cada chamada interceptada. É possível, além de monitorar, interferir no funcionamento dos processo, por exemplo, interceptando as funções de abertura de arquivo e, caso o acesso não seja autorizado, abortar a operação.

Uma vez que o funcionamento do *ptrace* ocorre no nível do núcleo do sistema, não é possível interceptar funções de alto nível. Quando a interceptação de funções de diversas bibliotecas

é necessária, os recursos do carregador dinâmico de bibliotecas do sistema – *ld.so* – podem ser empregadas. Utilizando variáveis de ambiente *LD_PRELOAD* e *LD_LIBRARY_PATH*, pode-se interceptar as chamadas de funções de bibliotecas compartilhadas de maneira transparente, monitorando ou alterando o seu funcionamento [69]. Como exemplo dessa abordagem, é possível citar o *GridBox* [74] para ambientes *Linux*.

Outra abordagem de monitoramento e controle da execução de aplicações consiste na utilização de um depurador, como, por exemplo, *gdb*, presente nos ambientes UNIX. Com a ajuda da ferramenta, é possível monitorar diversos *breakpoints* na execução de aplicações, observar o comportamento de funções, analisar o conteúdo da pilha e registradores.

Como exemplo de ferramentas de monitoramento contínua pode-se citar [70] e [199]. A primeira ferramenta, conhecida como *StatMonitor*, monitora o funcionamento de aplicações no nível do usuário, coletando informações sobre seu funcionamento periodicamente [135]. A segunda abordagem utiliza um *kernel* Linux modificado para monitorar aplicações de maneira transparente.

3.2.4 Ferramentas para avaliação de aplicações MPI

Um dos mecanismos mais comuns para programação paralela é a tecnologia de passagem de mensagens – *Message Passing Interface (MPI)* [33, 214]. Esse mecanismo oferece recursos de troca de mensagens entre diversos computadores do sistema, possibilitando transmissão síncrona e assíncrona de dados entre os *EPs*.

Entre diversas implementações de *MPI*, pode-se citar *MPICH* [102, 103] e *LAM-MPI* [33, 214], ambas amplamente utilizadas e disponíveis livremente. Como exemplo de implementação proprietária de *MPI* pode-se citar *Intel MPI Library* (anteriormente denominado *Vampir*) [178].

Inicialmente, *MPI* foi concebido para ser utilizado em ambientes de *cluster*, com configurações homogêneas e sem suporte às possíveis falhas nos *EPs*. Entretanto, com o crescimento do uso de ambientes distribuídos, isso tornou-se uma desvantagem clara. Dessa maneira, novas versões de *MPI* foram projetadas, oferecendo suporte para ambientes heterogêneos distribuídos, tais como *grids* computacionais [235].

Uma vez que a tecnologia *MPI* é implementada na forma de uma biblioteca de funções, as opções de depuração e monitoramento das aplicações são limitadas, sendo necessária a utilização de ferramentas ou bibliotecas específicas.

Uma das ferramentas para monitoramento, depuração e análise de aplicações paralelas amplamente utilizada é *Intel Trace Tools* (originalmente *Vampir*) [178], a qual permite tanto uma avaliação contínua de aplicações *MPI* quanto baseada em eventos. A ferramenta oferece suporte para execução controlada, *tracing* das aplicações, controle de fluxo e visualização dos parâmetros de comunicação entre os *EPs* durante a execução, sendo que é dividida em duas aplicações: *Intel Trace Collector*, utilizado para a coleta *on-line* das informações referentes à execução da aplicação e *Intel Trace Analyser* que analisa as informações adquiridas.

O monitoramento e extração do comportamento de aplicações são realizados por meio de uma biblioteca compartilhada, dinamicamente ligada a uma aplicação *MPI*. Os dados adquiridos são armazenados em formato *STF (Structured Trace Format)*. A análise de dados pode ser feita com a utilização do *Intel Trace Analyser*, utilizando uma interface gráfica para identificar pontos críticos, visualizar a sequência e o fluxo de dados, e observar o comportamento de aplicações em diversos *EPs*. Além disso, é possível instrumentar o código fonte de uma aplicação

MPI utilizando funções específicas da ferramenta, as quais definem, de maneira precisa, os dados a serem capturados.

Uma abordagem similar é utilizada no sistema *MPE (Multi Processing Environment)*, composto por um conjunto de bibliotecas, aplicações e ferramentas gráficas para análise de desempenho de aplicações *MPI*. O sistema utiliza abordagem baseada em eventos, oferecendo um conjunto de bibliotecas para *tracing* de eventos *MPI*, similares ao *Intel Trace Tools*, junto com um visualizador gráfico de eventos. O ambiente é disponível livremente.

Uma outra biblioteca que oferece funcionalidades para instrumentação de código *MPI* é *MPICL*. Essa biblioteca oferece funções de instrumentação de aplicações *MPI* e encontra-se disponível para diversas arquiteturas paralelas. Entretanto, é necessária instrumentação manual de código fonte.

Outra ferramenta que permite avaliar aplicações *MPI* é a *mpiP*. Ela é composta por uma biblioteca de análise estatística de eventos de aplicações distribuídas (*profiling*).

Além disso, pode-se citar a aplicação *Xmpi* que permite visualizar eventos e fluxo de dados entre os *EPs* para aplicações *MPI*. Aplicações devem utilizar arquivos *.trace*, gerados automaticamente em sua execução.

Entretanto, a grande maioria das ferramentas de avaliação de *MPI* existentes é limitada, pois não oferecem suporte completo para avaliação automática de aplicações sem a necessidade de alteração e instrumentação de código fonte. Para isso, uma nova técnica de avaliação *on-line* transparente de aplicações distribuídas foi proposta em [72], utilizando o conceito de interceptação dinâmica de chamadas do sistema e baseado no projeto *GridBox* [74].

3.3 Aplicações

O comportamento extraído de processos pode ser classificado e analisado segundo técnicas estatísticas [65, 85], histórico de execuções [99, 209], algoritmos de avaliação *on-line* de comportamento [9, 205], extração e classificação do comportamento para prever futuras ações das aplicações [198, 199].

Em [65], uma abordagem estatística é proposta para prever os requisitos de CPU, entrada e saída e memória de aplicações, através da utilização de agrupamento (*clustering*) estatístico (utilizando o algoritmo de *k*-médias) e modelo de cadeias de Markov para identificar regiões de aplicações com maior uso de recursos.

Uma abordagem similar é utilizada em [85], onde os autores executam repetidamente uma mesma aplicação e analisam variações de seu comportamento. Uma baixa variação entre execuções sucessivas é observada no trabalho, o que permitiu utilizar as características funcionais de aplicações para determinar seus comportamentos sem a necessidade de cooperação explícita do usuário.

Outros trabalhos [99, 209] classificam de maneira estática o comportamento de processos segundo seus históricos de execuções, determinando valores médios de uso de CPU, memória e mecanismos de entrada e saída para cada grupo. Os valores derivados são utilizados, posteriormente, para prever o comportamento de aplicações.

Estratégias de predição *on-line* do comportamento de processos utilizando o modelo de Bayes [20] e lógica *fuzzy* são apresentadas em [205]. Uma abordagem similar é utilizada em [9] e [54], onde informações coletadas durante a execução são utilizadas para controlar o mecanismo de balanceamento de carga.

Uma abordagem que utiliza técnicas de inteligência artificial para classificação e predição do comportamento é apresentada em [198, 199, 172]. Nesses trabalhos o comportamento de aplicações é automaticamente extraído e classificado, por meio de redes neurais, de acordo com a utilização de recursos computacionais. A avaliação do comportamento histórico de aplicações é utilizada pelos autores para otimizar as políticas de balanceamento de carga [200, 201, 202, 171].

Uma ferramenta para análise dos padrões de acesso nos mecanismos de memória compartilhada distribuída é proposta em [82]. A ferramenta permite a geração de *traces* e visualização do funcionamento da memória compartilhada distribuída, identificando acessos, possíveis problemas de desempenho e fluxo de comunicação entre elementos de processamento. O sistema utiliza a plataforma *Java* com espaço global de endereçamento distribuído, baseado em um sistema próprio de memória compartilhada, utilizando recursos da plataforma *Java* e oferecendo objetos com suporte a escrita múltipla. A ferramenta permite identificar sincronizações de acesso à memória compartilhada, leituras e escritas em objetos distribuídos e migrações de objetos. Experimentos comprovam uma queda de desempenho em torno de 2% causada pela coleta de informações (*tracing*).

Uma abordagem que utiliza as técnicas de classificação e predição do comportamento de processos em sistemas distribuídos é apresentada em [72]. No trabalho, *NAS Benchmark* é avaliado utilizando redes neurais e modelos estatísticos, demonstrando que as abordagens que misturam estas técnicas apresentam uma eficiência maior de que as abordagens individuais.

3.4 Considerações finais

Neste capítulo foram apresentadas as principais técnicas de extração de comportamento de aplicações. O comportamento pode ser extraído através da avaliação do código fonte dos mesmos ou utilizando uma das técnicas de extração *on-line* do comportamento. Ferramentas para ambas as abordagens foram apresentadas.

Capítulo 4

Classificação de padrões

4.1 Considerações iniciais

Classificação de padrões é um processo estatístico que visa agrupar padrões similares em diversas classes, ou *clusters*, de acordo com as suas características.

Enquanto é possível utilizar dados extraídos do comportamento de processos sem o processo de classificação, tal abordagem resulta em uma grande quantidade de variáveis repetitivas ou similares, o que dificulta a sua análise. Neste caso, é possível empregar técnicas de classificação ou agrupamento com o objetivo de reduzir a dimensionalidade de dados. Dessa maneira, é possível agrupar comportamentos similares e realizar predições sobre os pontos mais relevantes.

Diversas técnicas de classificação podem ser empregadas para a classificação de padrões, como as abordagens estatísticas e baseadas em redes neurais.

Entre as abordagens estatísticas, pode-se citar o modelo de *k*-médias, *c*-médias, clusterização *QT* e *Support Vector Machines*. As baseadas em redes neurais são mais variadas, incluindo mapas de Kohönnen (também conhecido como modelo *SOM*), redes neurais auto-expansíveis, tais como *GCS*, *GNG*, *GSOM*, *RCE*, *GWR* e *SONDE*, e outros modelos como redes *RBF*, *ART* e *ICA*.

O objetivo deste capítulo é a apresentação e discussão das principais técnicas de classificação de padrões descritas na literatura. Enquanto o foco do trabalho é a sua aplicação em classificação do comportamento de processos distribuídos, tais técnicas podem ser aplicadas em outras áreas de computação.

As técnicas apresentadas são classificadas de acordo com a abordagem utilizada, sendo que as baseadas em modelos estatísticos e estocásticos são apresentados na seção 4.2, e as baseadas em redes neurais na seção 4.3. A seção 4.4 sumariza as técnicas apresentadas e conclui este capítulo.

4.2 Técnicas estocásticas

Entre diversas técnicas utilizadas para classificar o comportamento de processos é possível destacar alguns modelos estatísticos, tais como os modelos de *k*-médias e *SVM*.

Esta seção tem como o objetivo descrever tais técnicas e suas aplicações para a classificação do comportamento de processos.

4.2.1 Modelo de k -médias

O modelo de k -médias é um modelo de clusterização iterativa, que visa agrupar os objetos em k partições [31, 84, 196, 4, 53]. O modelo tem como o objetivo a minimização do erro quadrático médio, ou a variância total *intra-cluster*, conforme a equação 4.1, onde existem k clusters S_i , sendo $i = 1, 2, \dots, k$, e μ_i representando o centróide, ou a média central de todos os padrões de entrada $x_j \in S_i$.

$$V = \sum_{i=1}^k \sum_{x_j \in S_i} |x_j - \mu_i|^2 \quad (4.1)$$

Diversos algoritmos foram propostos para a classificação de acordo com o modelo, sendo mais conhecido o de Lloyd [132], descrito a seguir:

1. Particionar o espaço de entrada em k clusters iniciais. Esse particionamento pode ser feito tanto da forma aleatória quanto por meio de utilização de uma heurística;
2. Para cada amostra:
 - (a) Encontrar o centróide, ou o ponto médio, mais próximo;
 - (b) Atribuir a amostra a cluster correspondente;
 - (c) Recalcular o centróide do cluster;
3. Repetir o algoritmo até os valores dos centróides dos clusters permanecerem constantes.

Um exemplo de classificação realizada pelo modelo de k -médias é demonstrado na figura 4.1, apresentando o resultado de classificação das entradas em um conjunto de clusters definidos.

O modelo de k -médias encontra o mínimo local, não necessariamente convergindo a uma solução ótima, sendo que o resultado final é influenciado pela configuração inicial dos clusters e amostras. A grande vantagem desta técnica, entretanto, é o seu baixo custo computacional, o que possibilita executar o procedimento repetitivamente até conseguir resultado adequado.

4.2.2 Clusterização fuzzy de c -médias

A técnica de clusterização fuzzy de c -médias é similar ao modelo de k -médias, com a diferença de que, ao invés de pertencer a um único cluster, para cada amostra é calculado um grau de semelhança a todos os clusters presentes no sistema [163, 127, 185, 161]. Dessa forma, cada ponto x tem um coeficiente $u_k(x)$, que determina o seu grau de semelhança com o cluster k . Geralmente, a soma desses coeficientes é igual a 1, conforme a equação 4.2, sendo N o número de clusters.

$$\sum_{k=1}^N u_k(x) = 1 \quad (4.2)$$

Neste modelo, o centróide é calculado como sendo a média de todos os pontos presentes no sistema, de acordo com a sua semelhança ao cluster atual, conforme demonstrado na equação 4.3, onde c_k é centróide do cluster k e m é constante de normalização.

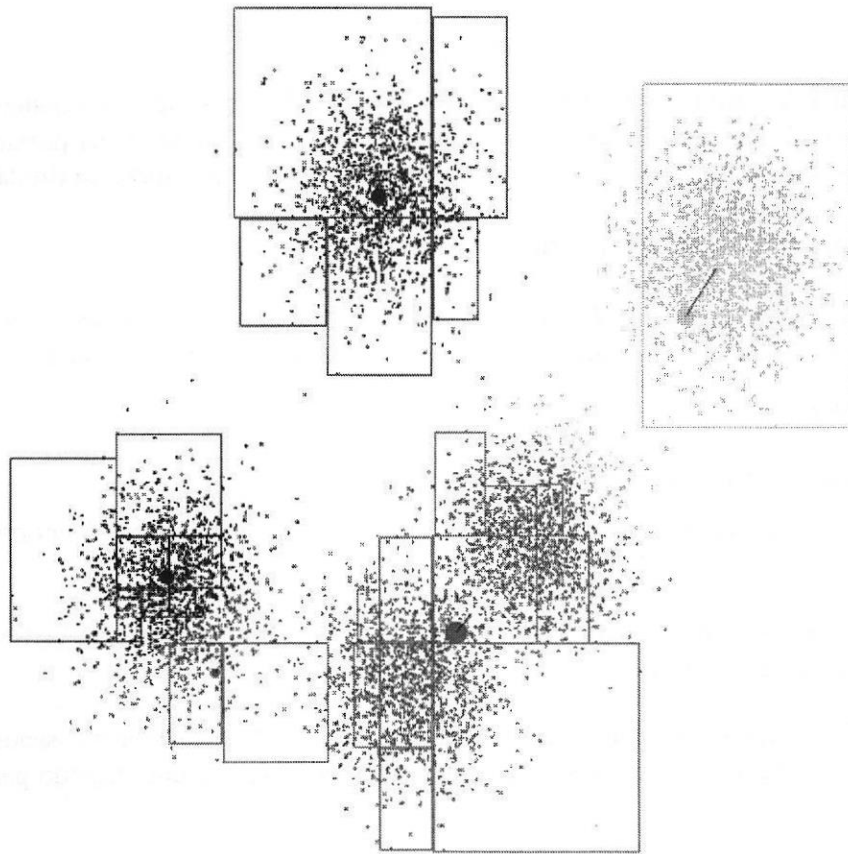


Figura 4.1: Classificação de acordo com o modelo de k -médias

$$c_k = \frac{\sum_x u_k(x)^m x}{\sum_x u_k(x)^m} \quad (4.3)$$

O grau de similaridade de uma amostra à cada *cluster* é determinado através da equação 4.4, onde $u_k(x)$ representa a semelhança do ponto x em relação ao *cluster* k .

$$u_k(x) = \frac{1}{d_c(k, x)^m} \quad (4.4)$$

Os coeficientes são normalizados, utilizando um coeficiente real $m > 1$, para satisfazer as equações 4.5 e 4.6, sendo m uma constante *fuzzy* de normalização e d a distância entre os padrões.

$$\sum_j^k u_k(x) = 1 \quad (4.5)$$

$$u_k(x) = \frac{1}{\sum_j \left(\frac{d(c_k, x)}{d(c_j, x)} \right)^{\frac{2}{m-1}}} \quad (4.6)$$

Para m igual a 2, a equação 4.6 resulta numa normalização linear dos coeficientes, onde a soma resulta em 1. Quando o valor de m aproxima-se a 1, os pontos mais próximos do centro do *cluster* recebem pesos maiores, e o comportamento do modelo torna-se similar ao de k -médias.

O funcionamento do algoritmo de c -médias é descrito a seguir:

1. Particionar o espaço de entrada em k *clusters* iniciais. Este particionamento pode ser feito tanto da forma aleatória quanto por meio de utilização de uma heurística.
2. Para cada amostra:
 - (a) Calcular o valor do centróide de cada *cluster*;
 - (b) Para cada amostra, calcular os coeficientes correspondentes aos *clusters* presentes no sistema;
3. Repetir o algoritmo até que a variação dos coeficientes entre duas iterações consecutivas do algoritmos seja inferior a ϵ .

O modelo de c -médias visa minimizar a variância *intra-clusters*. Entretanto, sendo similar ao modelo de k -médias, ele encontra mínimos locais, nem sempre convergindo para uma solução ótima.

4.2.3 Algoritmo de clusterização *QT*

Algoritmo de clusterização *QT* (*Quality Threshold*) [116, 129] é um método de agrupamento de dados alternativo, proposto inicialmente para a clusterização de genes do cromossomo humano. Enquanto ele requer maior poder computacional para atingir o resultado, o seu funcionamento não depende da pré-determinação do número de *clusters*, sempre retornando o mínimo global da solução.

O algoritmo de clusterização *QT* é apresentado a seguir:

1. Inicialmente, é necessário escolher o diâmetro máximo para cada *cluster*;
2. Para cada amostra, encontrar o *cluster*-candidato mais apropriado, considerando a distância do ponto atual até o ponto mais próximo do *cluster*, repetindo esse procedimento para todos os *clusters* que estão dentro do limite pré-estabelecido pelo diâmetro do passo 1;
3. Salvar o *cluster*-candidato que tem o maior número de amostras como sendo o primeiro *cluster* verdadeiro, removendo todos os pontos que pertencem a ele das iterações futuras;
4. Repetir o algoritmo com o número de pontos reduzidos, até que todas as amostras sejam alocados nos *clusters*.

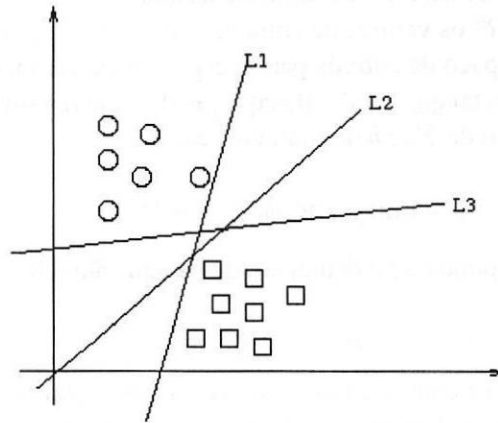


Figura 4.2: SVM – separação entre as classes

O valor de proximidade entre uma amostra e um *cluster* é calculado considerando a distância máxima entre ela e todos os pontos pertencentes ao *cluster*.

O modelo resulta em uma clusterização mais precisa, uma vez que um número adequado de *clusters* é alocado para cada caso. O custo computacional do modelo, entretanto, é significativamente maior de que o do modelo de *k*-médias e *c*-médias.

As técnicas de classificação iterativa também podem ser utilizadas para a predição de padrões, conforme demonstrado em [65], onde o modelo de *k*-médias é utilizado para identificar e prever diversos estados de execução de processos.

4.2.4 Classificação linear: SVM (Support Vector Machines)

SVM (Support Vector Machines) é uma técnica de aprendizagem linear supervisionada, voltada para classificação e regressão de padrões. Uma característica especial desta família de técnicas consiste no fato delas minimizarem o erro empírico de classificação, maximizando ao mesmo tempo a margem geométrica de erro. Com isso, essas técnicas são também conhecidas como classificadores de margem máxima (*maximum margin classifiers*) [197, 221, 240, 210].

A técnica consiste em mapeamento de vetores de entrada em um espaço amostral com dimensão superior, por meio de construção do hiper-plano máximo de separação. Para isso, dois hiper-planos auxiliares são construídos, de uma forma que o hiper-plano de separação resultante maximize a distância entre eles. É assumido que, quanto maior for a margem de distância entre os planos, menor é o erro de generalização do classificador.

A técnica de SVM classifica os padrões de entrada representados por um vetor de pesos ρ -dimensional em diversas classes, com o objetivo de separar tais classes com um hiper-plano de dimensão $\rho - 1$. Além disso, a técnica visa encontrar um hiper-plano que possibilita a maior separação entre as classes – ou seja, encontrar um cuja distância até o padrão atual de entrada seja máxima, conhecido como hiper-plano de margem máxima (ou *maximum-margin hyper-plane*). Como é possível ver na figura 4.2, enquanto três hiper-planos podem ser construídos para os valores de entrada considerados, somente um resulta em separação máxima entre as classes.

O algoritmo de *SVM* pode ser descrito da seguinte forma:

Considerando $X \in R_0 \subseteq \mathbb{R}^n$ os vetores de entrada, $y \in \{-1, +1\}$ os eixos, e $\phi : R_0 \rightarrow F$ a função de mapeamento do espaço de entrada para o espaço de apresentação, o algoritmo visa encontrar um hiper-plano (w, b) tal que o valor da equação 4.7 seja máximo, sendo γ a margem, vetor w com a mesma dimensão de F , e b um número real.

$$\gamma = \min_i y_i \{ \langle w, \phi(X_i) \rangle - b \} \quad (4.7)$$

A função de decisão correspondente é demonstrada na equação 4.8.

$$f(X) = \text{sign}(\langle w, \phi(X) \rangle - b) \quad (4.8)$$

O mínimo da função $f(X)$ ocorre quando a equação 4.9 é satisfeita, sendo que α_i são números reais positivos que maximizam a equação 4.10 e satisfazem a equação 4.11.

$$w = \sum_i \alpha_i y_i \phi(X_i) \quad (4.9)$$

$$\sum_i \alpha_i - \sum_{ij} \alpha_i \alpha_j y_i y_j \langle \phi(X_i), \phi(X_j) \rangle \quad (4.10)$$

$$\sum \alpha_i y_i = 0, \alpha_i > 0 \quad (4.11)$$

Desta forma, é possível representar a função de decisão através da equação 4.12.

$$f(X) = \text{sign}(\sum_i \alpha_i y_i \langle \phi(X_i), \phi(X) \rangle - b) \quad (4.12)$$

A partir da equação 4.12 observa-se que α_i , associado ao ponto de treinamento X_i , expressa a importância daquele ponto no resultado final da função de decisão.

É importante observar que somente um sub-conjunto de pontos é associado a α_i não-nulos. Tais pontos são denominados de vetores de suporte (*support vectors*, dando nome à técnica de *SVM*), e são os pontos mais próximos ao hiper-plano de separação.

Dessa forma, a técnica possibilita a classificação de padrões de acesso, separando-os de acordo com os vetores de suporte do hiper-plano determinados. A técnica pode ser utilizada para predição de padrões numa sequência longa, conforme demonstrado em [117].

4.3 Técnicas baseadas em redes neurais

Redes neurais artificiais são sistemas computacionais baseados em ligações, que visam simular a estrutura do cérebro humano. Uma rede é composta por um conjunto de elementos de processamento ("neurônios", "nós" ou "unidades"), interligados por meio de conexões com pesos associados, que se comunicam através de troca de sinais [155]. O funcionamento da rede neural é relativamente simples, sendo que cada neurônio é responsável apenas por receber sinais de seus vizinhos ou fontes externas, determinar o sinal de saída em função dos dados recebidos e o propagar para outras unidades de processamento, conforme ilustrado na figura 4.3 [155].

Geralmente, assume-se que cada unidade de processamento oferece uma contribuição aditiva para o neurônio à qual ela é conectada, calculada em função do peso associado à conexão entre eles. Dessa forma, o valor total de entrada s_k de um neurônio k pode ser representado

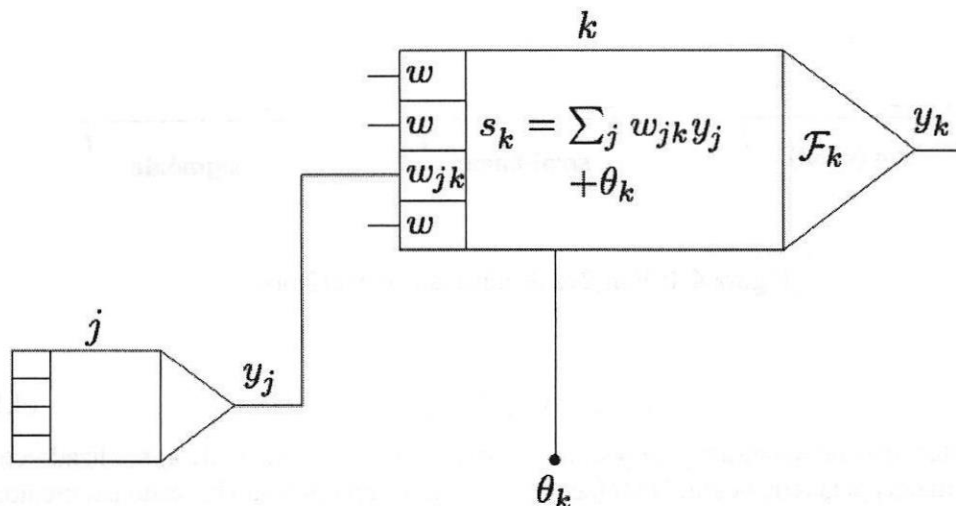


Figura 4.3: Componentes de uma rede neural

como uma soma ponderada dos valores de saída y_j dos neurônios j ajustada em função do peso w_{jk} , associado à conexão entre neurônio j e k , mais um valor inicial $\theta_k(t)$ do neurônio k , conforme a equação 4.13.

$$s_k(t) = \sum_j w_{jk}(t) y_j(t) + \theta_k(t) \quad (4.13)$$

A contribuição de um neurônio, determinada pelo peso da conexão w_j , pode ser negativa, utilizada para *inibição* de um neurônio, ou positiva, utilizada para a sua *excitação*.

Os pesos das conexões podem ser ajustados dinamicamente para indicar conexões mais relevantes durante um processo de treinamento ou aprendizado. O treinamento de uma rede neural pode ser feito de maneira supervisionada, ou *associativa* [39], quando a rede é alimentada com valores pré-determinados de entrada e saída para possibilitar reconhecimento de uma série de padrões de entrada. Outro tipo de treinamento é conhecido como treinamento não-supervisionado, ou *auto-organizável* [155], na qual a rede se adapta dinamicamente ao seu conteúdo sem nenhum conhecimento prévio sobre os padrões de entrada.

O aprendizado de rede é feito de acordo com uma regra de aprendizado. Geralmente, o modelo de aprendizado *Hebbiano* [113] é aplicado, no qual afirma-se que, caso duas unidades de processamento j e k estiverem ativas simultaneamente, sua conexão deve ser reforçada, ajustando o peso dessa conexão. O ajuste do peso pode ser feito de maneira direta, ajustando o peso da conexão de acordo com a equação 4.14, onde τ é uma constante que representa a taxa de aprendizado da rede (eq. 4.14):

$$\delta w_{jk} = \tau y_j y_k \quad (4.14)$$

Uma outra maneira de ajustar o peso de uma conexão consiste na técnica de aproximação, que visa adaptar gradativamente a diferença entre o peso atual da conexão y_k e o peso desejado d_k , informado explicitamente, conforme a equação 4.15.

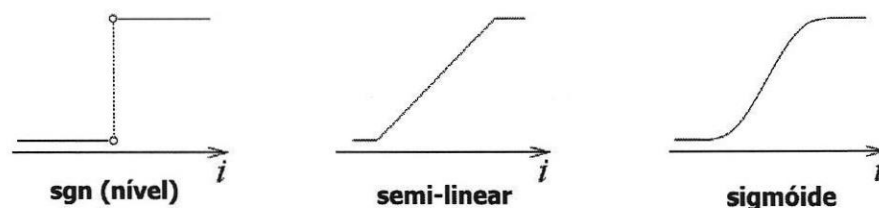


Figura 4.4: Funções de ativação de neurônios

$$\delta w_{jk} = \tau y_j (d_k - y_k) \quad (4.15)$$

Outras técnicas de aprendizagem podem ser aplicadas, tais como as de aprendizado competitivo, utilizada pela rede neural *SOM* (seção 4.3.1), as de aprendizado baseado em memória, utilizada pela rede *RSOM* (seção 5.6.1), as de aprendizado de Boltzmann, utilizada pela rede de Hopfield (seção 5.5.2) entre outras.

Em redes com múltiplas conexões entre os neurônios, cada elemento tem um valor de ativação $y_k(t)$, usado para determinar os neurônios mais adequados para a propagação do sinal calculado pelo neurônio atual. A escolha do próximo neurônio $y_k(t+1)$ é feita por uma *função de ativação* $F_k(t)$ com base no valor total de entrada $s_k(t)$ do neurônio k (eq. 4.16).

$$y_k(t+1) = F_k(t_k(t), s_k(t)) \quad (4.16)$$

O valor de ativação y_k do neurônio k é controlado pelo valor de nível (*threshold*) s e pode ser calculado por diversas funções, tais como função de corte (*sgn*), semi-linear ou sigmóide (equação 4.17), conforme a figura 4.4 [155].

$$y_k = F(s_k) = \frac{1}{1 + e^{-s_k}} \quad (4.17)$$

Uma rede neural geralmente é composta por uma série de camadas de neurônios, conforme a figura 4.5, com uma camada de entrada cujo objetivo é receber dados de entrada, uma camada de saída e uma série de camadas escondidas utilizadas para o processamento e propagação de sinais.

De acordo com o padrão de conexões entre as camadas, é possível classificar as redes neurais em redes *feed-forward*, onde os dados somente são propagados da camada de entrada até a camada de saída, e redes *recorrentes*, que possibilitam conexões de *feedback* para ajustar dinamicamente a configuração da rede. Como exemplos de redes *feed-forward* é possível citar redes *Adaline* e *Perceptron* [155], e como exemplos de redes recorrentes pode-se citar redes *SOM* [143] e redes de Hopfield [120].

Atualmente existe uma grande variedade de redes neurais com vasta aplicação em diversas áreas computacionais. No caso de sistemas distribuídos, é possível representar um ambiente distribuído como sendo composto por diversos neurônios, cada um representando um elemento de processamento. Nesse caso, os parâmetros dos neurônios representariam o poder de processamento, memória disponível, latência de acessos, etc. Dessa forma, a rede neural pode ser utilizada para solucionar diversos problemas, tais como a distribuição de dados pelo ambi-

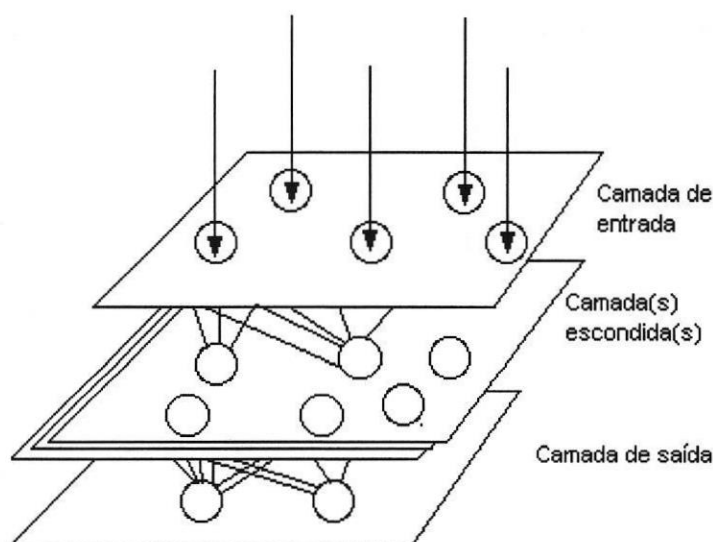


Figura 4.5: Topologia de rede neural

ente em função de parâmetros de recursos computacionais [170] ou antecipação de acessos aos dados [72].

A seguir, as principais redes neurais usadas para classificação de padrões são apresentadas.

4.3.1 Mapas auto-organizáveis – SOM

Os mapas auto-organizáveis (*SOM – Self-Organizing Maps*) foram introduzidos por Kohonen [143] com o objetivo de resolver o problema de classificação de dados de entrada de acordo com o nível de similaridades [133, 143]. *SOM* utiliza treinamento não-supervisionado, onde o neurônio que mais se adequa a um determinado padrão de entrada é utilizado para classificar padrões de entrada similares. Um exemplo de classificação feita por *SOM* é apresentado na figura 4.6 [166].

Supondo que a figura 4.6 representa os processos executados em um ambiente distribuído, pode-se ver que processos com características similares (por exemplo, uso do processador, memória e taxa de transmissão) são agrupados em *clusters* com características similares.

SOM representa os dados de entrada como pontos, ou nós, em um mapa. Após o mapeamento inicial de dados, o algoritmo *SOM* é aplicado sobre um vetor de entrada de dimensões arbitrárias contendo todos os nós do mapa com o objetivo de agrupar vetores semelhantes, através do processo *winner-take-all*. Nesse processo, o neurônio com vetor de peso mais similar ao vetor de entrada é declarado vencedor e tem seu peso ajustado, visando torná-lo mais próximo ao vetor de entrada. Além disso, para cada neurônio no mapa é determinada a sua vizinhança (conjunto de neurônios, cujo peso é próximo ao do neurônio vencedor), ajustando os pesos dos neurônios encontrados proporcionalmente à sua distância do vencedor. O processo é repetido durante um número arbitrário de ciclos.

O algoritmo pode ser descrito da seguinte maneira:

1. Inicializar a rede, representada como um mapa, conforme demonstrado na figura 4.6,

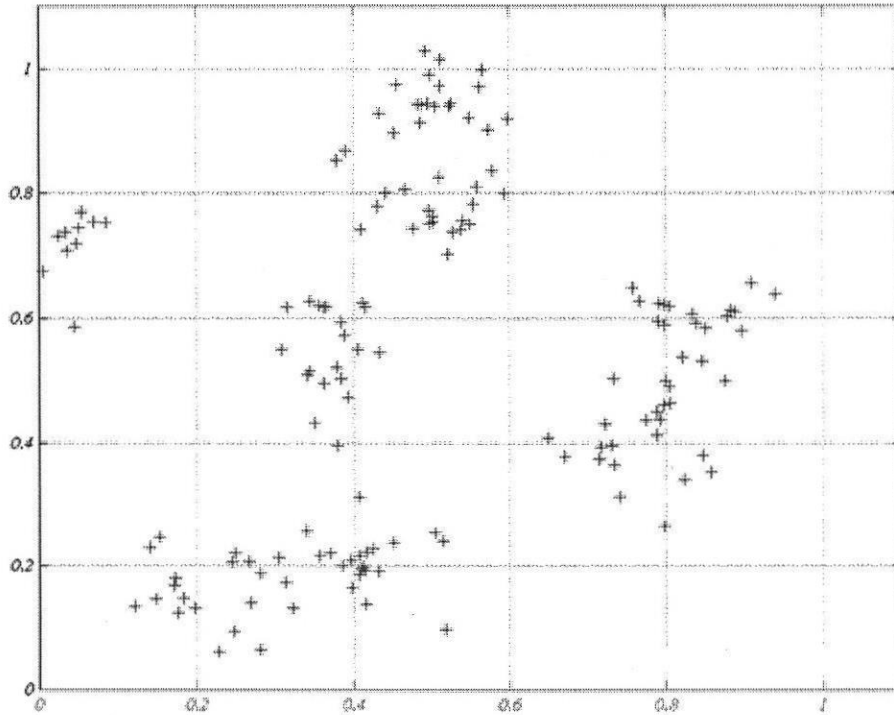


Figura 4.6: SOM: agrupamento de dados

com neurônios aleatórios;

2. Para cada vetor de entrada:

(a) Para cada neurônio presente no mapa:

- i. Determinar a distância Euclidiana D_E entre o neurônio encontrado q e o vetor de entrada p para calcular a sua semelhança por meio da equação 4.18, onde p_x representa o x -ésimo elemento do vetor de entrada e q_y representa o y -ésimo elemento do neurônio atual;

$$D_E = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^n (p_i - q_i)^2} \quad (4.18)$$

- ii. Salvar o neurônio j com peso w_j que apresentou a menor distância D do padrão de entrada x , satisfazendo a equação 4.19, o qual é denominado de *BMU* (*Best Matching Unit*).

$$D = \operatorname{argmin} \|x - w_j\| \quad (4.19)$$

(b) Determinar os neurônios em vizinhança do *BMU*;

(c) Para cada neurônio k vizinho do BMU :

- i. Atualizar os pesos do neurônio k , aproximando-os aos do BMU , conforme equação 4.20, onde t representa a iteração atual, Wv representa o vetor de pesos do neurônio k , $\omega(t)$ representa a distância entre o neurônio k e o BMU , $\alpha(t)$ representa a taxa de aprendizagem e $D(t)$ é a distância entre o neurônio k e o BMU .

$$Wv(t+1) = Wv(t) + \omega(t)\alpha(t)(D(t) - Wv(t)) \quad (4.20)$$

3. Repetir para todos os vetores restantes.

Considerando que para cada vetor de entrada é determinado um neurônio-vencedor (BMU) diferente, o algoritmo possibilita agrupar padrões em um número de *clusters* com características similares.

A variação do parâmetro $\alpha(t)$ possibilita controlar a taxa de aprendizado do algoritmo *SOM*. Com valores altos, pode-se obter uma taxa de aprendizagem rápida, enquanto valores baixos resultam em treinamento mais refinado.

O *SOM* apresenta dois modos de operação:

1. Durante o processo de *treinamento*, é construído um mapa, organizando os vetores de entrada através do processo competitivo *winner-take-all*.
2. Durante o processo de classificação, ou *clustering*, os vetores de entrada são classificados de acordo com o mapa criado no passo anterior.

Nesse caso, apenas o passo 2.a do algoritmo é realizado, determinando o BMU para cada elemento de entrada. Após isso, o vetor de entrada é classificado no *cluster* correspondente ao neurônio BMU encontrado.

Redes *SOM* são aplicadas em diversas áreas, onde é evidente a necessidade de classificação e organização de grande número de informações. Como exemplos de aplicações de redes *SOM* é possível citar organizações de bases de dados [143], predição do uso de energia [159], classificação de dados de satélites [175] e processamento de imagens [7].

4.3.2 Redes neurais auto-expansíveis

Enquanto redes neurais classificadoras, tais como *SOM*, permitem classificação automática e não-supervisionada de dados, elas apresentam limitações relacionadas à necessidade de definição prévia de configuração, topologia, e parâmetros de treinamento da rede.

Com o objetivo de permitir uma classificação mais adaptativa, diversas redes neurais auto-expansíveis, ou seja, redes que possibilitam a criação de novos neurônios sob demanda, foram propostas, tais como *Cascade-Correlation Learning Architecture (CCLA)* [81, 213, 13], *Growing Cell Structure (GCS)* [156, 215], *Probabilistic GCS* [226, 34, 47], *Growing Neural Gas (GNG)* [156, 107, 91], *Growing Self-Organizing Maps* [5, 61], *RCE* [75, 236], *CLAM* [219], *Growing When Required (GWR)* [167] e *SONDE* [6].

A maioria das redes neurais auto-expansíveis são similares ao modelo de redes *SOM*. A seguir, as principais características delas serão discutidas.

4.3.3 *Cascade-Correlation Learning Architecture*

A rede neural *CCLA* é uma rede neural supervisionada, criada com o objetivo de otimizar o desempenho do processo de aprendizagem [81, 213, 13]. De maneira similar à outras redes neurais classificadoras, a *CCLA* é inicializada com uma estrutura mínima, sendo que novos neurônios são inseridos à medida que for necessário.

A principal diferença de *CCLA* em relação a outras redes classificadores consiste no fato dela empregar treinamento supervisionado, adicionando novos neurônios nas camadas escondidas da rede somente quando necessário. Novas unidades são inseridas na rede quando o erro de classificação não é reduzido durante iterações consecutivas do processo de treinamento. Neste caso, o seguinte algoritmo é executado:

1. Um novo neurônio-candidato é criado;
2. Os pesos do neurônio são treinados, visando maximizar a correlação entre a saída do neurônio e o erro residual de saída. Ao final do treinamento, este neurônio é adicionado à rede;
3. Ao ser incluído na rede, os pesos do neurônio são congelados, não permitindo modificações futuras nos seus valores.

Dessa maneira, a rede *CCLA* visa uma criação rápida de rede classificatória, criando neurônios adicionais dinamicamente.

4.3.4 *Growing Cell Structure*

A rede *GCS* (*Growing Cell Structure*), apresentada por Fritzke [91, 156, 215], foi uma das primeiras redes neurais auto-expansíveis não supervisionadas. A rede é baseada na *SOM*, sendo composta por superfícies k -dimensionais, sendo que o valor de k é geralmente definido como sendo $k = 2$. Com isso, as superfícies criadas são sempre triangulares.

Na rede *GCS*, um novo neurônio é inserido a cada λ iterações, sendo λ uma constante pré-definida, de forma a suportar o neurônio que acumulou o maior número de erros durante as iterações passadas. Esse processo continua até que um critério de parada seja satisfeito, seja ele um tamanho pré-determinado da rede, ou um valor mínimo para o erro acumulado.

Diversas variações da rede *GCS* foram propostas, visando otimizar seu funcionamento. Em [34], Burzevski e Mohan propuseram um novo mecanismo para tratar as mudanças da estrutura de rede que podem ocorrer devido à remoção de neurônios – uma vez que a rede é composta por uma série de superfícies k -dimensionais, a remoção de um neurônio pode modificar drasticamente a sua estrutura.

Uma das propriedades fundamentais da rede *GCS* é o fato dela manter a mesma topologia durante todo o processo de treinamento, aplicando uma divisão de neurônios existentes, criando novas ligações entre eles. Dessa forma, quando um elemento é removido da rede, o neurônio mais próximo a ele é dividido em dois, visando manter a mesma topologia.

De acordo com os autores, esse algoritmo oferece diversas vantagens em alguns casos, resultando em uma classificação mais precisa e eficiente, a custo de aumento significativo de número de neurônios na rede, resultando em uma topologia mais complexa da rede neural [34].

Uma extensão à rede, conhecida como *DCS* (*Dynamic Cell Structures*), foi proposta por Bruske e Sommer [3, 83]. Nessa extensão, os autores propõem a inserção de novos neurônios

na rede de uma forma a manter a topologia da rede, diferentemente da rede *GCS* original, que visa somente equalizar os valores de erro esperados em cada neurônio.

Uma abordagem diferente é proposta por Vlassis, conhecida como *Probabilistic GCS* [226, 34, 47]. O modelo proposto visa tornar a rede *GCS* probabilística, com o objetivo de melhorar o mecanismo de tratamento de entradas. Para isso, as entradas da rede são avaliadas por meio de técnicas estocásticas, visando determinar co-relações entre elas.

Finalmente, uma outra extensão para da rede *GCS* foi proposta por Cheng e Zell [47], alterando a estrutura básica do algoritmo de *GCS*. No trabalho, a rede foi modificada, permitindo com que mais de um neurônio pudesse ser adicionado simultaneamente. Dessa forma, dependendo da topologia atual, dois ou mais neurônios podem ser adicionados à rede simultaneamente, dependendo da posição do neurônio-vencedor atual, resultando em uma convergência mais rápida da rede, ao custo de criação de um número maior de neurônios.

Tanto a rede original *GCS*, quanto as suas extensões, são empregados em diversas áreas computacionais, porém a sua eficiência varia de acordo com os padrões de entrada avaliados.

4.3.5 Growing Neural Gas

A rede neural *Growing Neural Gas (GNG)*, também proposta por Fritzke [156, 107, 91], é similar à *GCS*. Na rede, os neurônios são adicionados a cada λ iterações, de forma similar à rede *GCS*, visando suportar o neurônio com maior erro acumulado. Entretanto, diferentemente de *GCS*, a estrutura topológica da rede não é mantida, sendo que as conexões são criadas entre dois elementos que apresentaram a maior atividade: para cada padrão de entrada.

Dessa forma, para cada entrada do sistema são selecionados dois neurônios-vencedores, cuja distância Euclidiana do padrão de entrada é menor. Tais neurônios são conhecidos como vencedores, uma conexão entre eles é criada e suas posições alteradas de forma com que os seus pesos sejam mais adequados ao valor de entrada. Além disso, as posições dos neurônios-vizinhos dos vencedores também são ajustadas.

Visando manter somente as conexões entre os neurônios mais significativos, as conexões que não foram atualizadas por um determinado número de iterações são removidas. Após λ iterações, o neurônio que acumulou a maior taxa de erros durante as execuções passadas do algoritmo é determinado, e um novo elemento é inserido na rede para diminuir a taxa de erros. O novo neurônio é posicionado entre o que apresentou a maior taxa de erro e o seu vizinho com a taxa de erros mais próxima. O algoritmo continua até atingir um critério de parada pré-estabelecido.

Diversos trabalhos comparam o desempenho das redes *GCS* e *GNG* [156, 107, 91], concluindo que a eficiência dessas redes varia de acordo com o padrão de entrada avaliado.

Visando melhorar o desempenho da rede *GNG* para padrões de entrada não-estacionários, uma modificação da rede foi proposta por Fritzke [92], denominada de *GNGU (Growing Neural Gas with Utility)*. Essa abordagem introduz o conceito da “utilidade” dos neurônios, calculada por meio da avaliação da possível variação do erro global do sistema, caso tal neurônio for removido. Portanto, para manter somente os dados mais relevantes, os neurônios que apresentam baixa utilidade são deletados, mantendo somente os mais adequados.

4.3.6 *Growing Self-Organizing Maps*

Uma outra abordagem é apresentada por Bauer e Villmann [5, 61], introduzindo uma extensão da rede neural *SOM* conhecida como *Growing Self-Organizing Map*. O algoritmo de treinamento é baseado no algoritmo original de *SOM*, com a diferença de que a cada λ iterações novos neurônios são acrescentados na rede. No lugar de inserir somente um elemento por vez, supondo que as conexões entre os neurônios serão estabelecidas automaticamente durante o treinamento, o algoritmo proposto adiciona uma série de neurônios simultaneamente, mantendo a topologia da rede.

O algoritmo visa manter a estrutura da rede inserindo um conjunto de neurônios de uma vez, de acordo com a topologia atual da rede. Ao adicionar novas entradas na rede, a vizinhança entre os neurônios atuais é mantida, considerando a correspondência topológica entre dados de entrada e a sua representação na rede.

4.3.7 *Restricted Coulomb Energy*

A rede neural *Restricted Coulomb Energy (RCE)* é uma rede não-supervisionada, que classifica padrões de entrada sem criar conexões entre os neurônios-vizinhos [75, 236]. A rede utiliza séries de vetores-protótipos para classes particulares de entrada.

Caso nenhum dos vetores existentes, cada um dos quais representa um *cluster*, é suficientemente próximo ao valor de entrada avaliado, uma nova classe é criada e inicializada com o valor de entrada atual. Com isso, não existem conexões entre os *clusters*, e os protótipos não são modificados após serem criados. Uma abordagem similar é empregada na família de redes neurais *ART (Adaptive Resonance Theory)*, apresentada na seção 4.3.10.

4.3.8 *Contextual Layered Associative Memory*

Uma abordagem diferente é proposta por Thacker e Mayhew [37, 220, 219], introduzindo a rede neural *CLAM (Contextual Layered Associative memory)*. A rede é composta por uma série de camadas interconectadas com um mecanismo de ressonância dentro de cada, utilizado para detectar similaridades entre os padrões de entrada.

Diferentemente de outras redes, que utilizam a abordagem *winner-takes-all* para escolher um único vencedor, *CLAM* classifica os padrões entre diversos grupos de neurônios, efetivamente cobrindo o espaço amostral dos padrões de entrada. Neurônios adicionais são acrescentados na rede quando a mesma detecta que uma região de entrada apresenta uma baixa densidade de neurônios.

4.3.9 *Growing When Required*

A rede neural *Growing When Required (GWR)*, proposta por Marsland *et al.* [167], é uma rede auto-organizável e auto-expansível, criada com o objetivo de suprir as deficiências das redes auto-expansíveis existentes, sendo amplamente utilizada para classificação de padrões e detecção de novidades.

A rede é composta por neurônios interconectados com vetores de pesos conectados com os seus vizinhos, sendo que os padrões de entrada similares são representados por meio de agrupamentos de neurônios. Tanto os nós da rede quanto as conexões são criados e destruídos durante o processo de aprendizagem dinamicamente.

Diferentemente de *GNG* e *GCS*, onde novos neurônios são inseridos na rede a cada λ iterações, *GWR* permite a criação de novos neurônios a qualquer momento. A rede utiliza aprendizado Hebbiano competitivo, e para cada padrão de entrada uma conexão entre os dois neurônios-vencedores é criado. À cada conexão é associado um parâmetro representando a sua idade, inicializado com zero, sendo que as idades das conexões estabelecidas são incrementadas a cada iteração, e as cuja idade excede a constante α_{max} são removidas. Da mesma forma, os neurônios sem nenhuma conexão também são removidos, por serem considerados mortos.

Diferentemente de *GNG*, onde novos neurônios são inseridos de forma a suportar neurônios com maior taxa de erro acumulada, na rede *GWR* eles são criados de acordo com a topologia da rede. Para adicionar um neurônio, a atividade dos neurônios presentes no sistema é calculada, sendo representada pela distância Euclidiana entre o vetor de pesos de neurônio e atual padrão de entrada. Neurônios com menor distância são escolhidos como vencedores.

Levando em consideração o fato de que neurônios recém-criados podem não ser treinados o suficientemente para realizar uma classificação adequada, a rede introduz um mecanismo para avaliar quantas vezes cada neurônio foi disparado. Geralmente, um contador simples é utilizado para calcular tal valor, sendo incrementado cada vez que o neurônio for considerado um vencedor.

Uma outra abordagem consiste na utilização de uma variável que decresce exponencialmente de 1 a 0, sendo que os neurônios recém-criados são inicializados com 1 e os neurônios disparados frequentemente tendem a ter seus valores próximos a 0. Essa abordagem é equivalente à anterior, porém possibilita a modificação das variáveis dos neurônios-vizinhos do vencedor, levando em consideração a distância entre eles. Além disso, o número de vezes que um neurônio foi disparado também pode ser levado em consideração durante o processo de aprendizagem, de uma forma que os disparados com maior frequência são treinados menos do que os que raramente ativados. Com isso, é possível otimizar a convergência da rede, evitando super-utilização de alguns neurônios a custo de outros, o que pode levar a uma classificação imprecisa.

Dessa forma, quando um novo padrão de entrada é processado pela rede, a atividade de todos os neurônios é calculada, escolhendo um vencedor. Se o valor que representa a sua atividade é próximo a 1, a rede considera que ele representa o padrão de entrada de forma adequada. Caso contrário, se a atividade da rede é inferior ao valor mínimo estabelecido, determinado por α_T , isso pode significar que o neurônio foi inserido na rede recentemente ou que nenhum elemento da rede representa o padrão de entrada adequadamente.

Se o neurônio foi criado recentemente, ele é treinado. Caso contrário, um novo neurônio é criado entre o vencedor e o padrão de entrada considerado, tendo seus pesos inicializados com valores médios dos pesos do neurônio-vencedor e o atual padrão de entrada.

Com isso, o comportamento da rede pode ser configurado por meio do parâmetro α_T , sendo que quando α_T é próximo a 1, a fase de treinamento cria um número maior de neurônios, resultando em classificação mais detalhada. Por outro lado, quando o valor de α_T é baixo, a rede resulta em uma classificação mais generalizada.

O algoritmo da rede *GWR* é descrito a seguir:

1. Considerando A o conjunto de neurônios presentes na rede, $C \in A \times A$ o conjunto das conexões entre eles, ω_n o vetor de pesos do neurônio n e ξ os valores de entrada, cujo espaço amostral é representado por $\rho(\xi)$, a rede é inicializada com dois neurônios n_1 e n_2 , escolhidos aleatoriamente de $\rho(\xi)$;

2. Para cada iteração do algoritmo:

- (a) Escolher um padrão de entrada ξ ;
- (b) Para cada neurônio i da rede, calcular a distância Euclidiana dele para o padrão de entrada atual $||\xi - \omega_i||$;
- (c) Selecionar os dois neurônios-vencedores, representados por $s, t \in A$, de acordo com as equações 4.21 e 4.22, onde ω_n é o vetor de pesos do neurônio n ;

$$s = \arg \min_{n \in A} ||\xi - \omega_n|| \quad (4.21)$$

$$t = \arg \min_{n \in A/\{s\}} ||\xi - \omega_n|| \quad (4.22)$$

- (d) Caso não exista conexão entre s e t , a conexão é criada entre eles, tendo a sua idade inicializada com 0. Caso contrário, a idade da conexão é zerada;
- (e) A atividade do neurônio-vencedor é calculada através da equação 4.23;

$$\alpha = \exp(-||\xi - \omega_s||) \quad (4.23)$$

- (f) Caso o valor da atividade α seja inferior ao valor de corte α_T , e o contador de disparos do neurônio seja inferior ao valor de corte h_T , um novo neurônio r é inserido entre os dois neurônios-vencedores (s e t), tendo o seu vetor de pesos ω_r inicializado de acordo com a equação 4.24.

$$\omega_r = (\omega_s + \xi)/2 \quad (4.24)$$

Após isso, conexões entre r e s são criadas, e conexão entre s e t é removida da rede;

- (g) Caso a criação de um novo neurônio não for necessária, as posições do neurônio-vencedor e dos seus vizinhos i são ajustadas de acordo com as equações 4.25 e 4.26, considerando $0 < \epsilon_n < \epsilon_b < 1$ e h_s como sendo o valor de corte de contagem de disparos para o neurônio s ;

$$\delta\omega_s = \epsilon_b \times h_s \times (\xi - \omega_s) \quad (4.25)$$

$$\delta\omega_i = \epsilon_n \times h_i \times (\xi - \omega_i) \quad (4.26)$$

- (h) A idade das conexões que levam a s é incrementada de acordo com a equação 4.27, sendo $a_{(s,i)}$ a idade da conexão entre os neurônios s e i ;

$$a_{(s,i)} = a_{(s,i)} + 1 \quad (4.27)$$

- (i) O contador de disparos do neurônio-vencedor s é decrementado, de acordo com a equação 4.28; e o dos seus vizinhos de acordo com a equação 4.29, sendo que $h_i(t)$ representa o valor do contador de disparos do neurônio i , h_0 é o valor inicial do

contador, $S(t)$ representa a constante de estimulação, geralmente sendo igual a 1, α_n , e α_b , θ_n e θ_b são parâmetros que controlam o comportamento da equação;

$$h_s(t) = h_0 - \frac{S(t)}{\alpha_b} (1 - e^{(-\alpha_b t / \theta_b)}) \quad (4.28)$$

$$h_s(t) = h_0 - \frac{S(t)}{\alpha_b} (1 - e^{(-\alpha_b t / \theta_b)}) \quad (4.29)$$

- (j) Após isso, existência de neurônios e conexões a serem removidos da rede é verificada. Os neurônios são removidos quando eles não são conectados a nenhum outro elemento da rede (i.e., são considerados neurônios mortos). As conexões, por sua vez, são removidos quando a sua idade excede a idade máxima pré-estabelecida;
- (k) O algoritmo é executado até atingir um critério da parada pré-estabelecido.

A rede *GWR* é caracterizada por se adaptar rapidamente às mudanças no padrão de entrada, sendo amplamente utilizada para detecção de novidades e classificação de padrões.

4.3.10 Adaptive Resonance Theory

ART (Adaptive Resonance Theory) é uma família de arquiteturas de redes neurais desenvolvida por Grossberg [40], caracterizadas por adotarem o modelo não-supervisionado de aprendizado e serem compostas por camadas de comparação e reconhecimento de padrões, um parâmetro de vigilância e um módulo de *reset*.

A camada de comparação e reconhecimento de padrões são responsáveis pela determinação de semelhanças entre os elementos do vetor de entrada e sua classificação. O parâmetro de vigilância é diretamente responsável pela precisão da rede neural – quanto maior for o valor desse parâmetro, maior é a precisão da classificação, ao custo de generalização. Em outras palavras, quanto maior for o valor do parâmetro, mais *clusters*, ou classes, independentes serão encontrados pelo algoritmo durante o processo de classificação de dados. Finalmente, o módulo de *reset* é utilizado para controlar o algoritmo, verificando a precisão de classificação em função do parâmetro de vigilância.

Um exemplo de classificação feita por *ART* pode ser visto na figura 4.7 [166]. Como pode-se ver na figura, os dados foram classificados em 5 *clusters* independentes, cada um com seu conjunto de características. A distância média entre *clusters* independentes é denominada distância *inter-clusters*, e a distância média entre os elementos de um mesmo *cluster* é denominada distância *intra-cluster* [172]. Esses parâmetros determinam o grau de independência entre os dados classificados.

A classificação de dados é feita de acordo com o algoritmo a seguir:

1. A camada de comparação recebe um vetor de entrada e o transmite para a camada de reconhecimento, onde o neurônio mais similar (i.e., cujo conjunto de pesos é mais próximo ao vetor de entrada) é encontrado. Cada neurônio da camada de reconhecimento alimenta um sinal negativo, proporcional à semelhança do neurônio encontrado com o padrão de entrada, para outros neurônios da camada de reconhecimento, conseqüentemente inibindo a sua saída. Dessa forma, a camada de reconhecimento apresenta inibição lateral, permitindo com que os seus neurônios representem diversas categorias nas quais os dados de entrada são classificados;

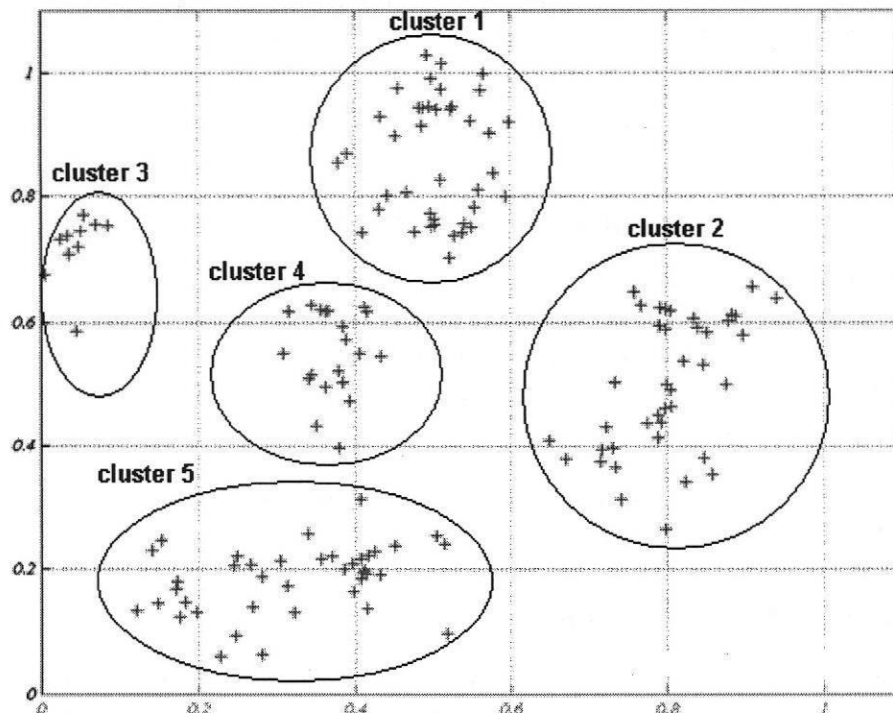


Figura 4.7: ART: classificação de dados

2. Após a classificação do vetor de entrada, o módulo de reinício (*reset*) compara o valor do padrão reconhecido ao parâmetro de vigilância. Caso o parâmetro de vigilância for satisfeito, o treinamento é iniciado. Caso contrário, o neurônio reconhecido é inibido até o processamento do próximo vetor de entrada. Nesse caso, treinamento começa somente após completar o procedimento de busca;
3. No procedimento de busca, os neurônios da camada de reconhecimento são desabilitados um por um, até satisfazer o parâmetro de vigilância. Caso nenhum neurônio reconhecido satisfaça o parâmetro de vigilância, um novo neurônio é criado para o padrão de entrada atual. Esse neurônio é considerado satisfatório e é ajustado em direção ao vetor de entrada.

Existem dois principais métodos de treinamento de redes neurais *ART* – lento e rápido. No método lento, o nível de treinamento do neurônio de reconhecimento é ajustado em direção ao vetor de entrada através de funções diferenciais e, conseqüentemente, seu desempenho depende do tamanho do vetor de entrada. No método de treinamento rápido, são utilizadas equações matemáticas para calcular os ajustes de pesos dos neurônios, utilizando valores binários.

A família *ART* é composta de várias redes. A versão mais simples da rede é conhecida como *ART-1* [39], sendo que o seu funcionamento é limitado pelo fato de somente aceitar valores binários no vetor de entrada.

Uma extensão da rede *ART-1* é conhecida como *ART-2* [40], caracterizada pela capacidade

de suportar valores de entrada contínuos. Entretanto, devido ao baixo desempenho dessa rede, uma modificação foi proposta, conhecida como *ART-2A* [112, 172], descrita de forma detalhada a seguir.

Visando estender a rede *ART-2*, a rede *ART-3* foi proposta [41]. Essa rede é caracterizada pela possibilidade de inibição parcial de neurônios, através de um mecanismo de regulação neuro-transmitida.

Com o objetivo de melhorar a generalização da rede, foi criada uma modificação denominada de *Fuzzy ART* [44]. A rede emprega lógica *fuzzy* no algoritmo de reconhecimento de padrões para reduzir o número de *clusters* criados no processo de classificação.

Rede *ARTMAP* [43], também conhecida como *Predictive ART*, combina as redes *ART-1* e *ART-2* em uma estrutura de aprendizagem supervisionada. Nesse caso, os dados reconhecidos são utilizados para adaptar o parâmetro de vigilância de maneira recorrente. Uma extensão dessa rede é conhecida como *Fuzzy ARTMAP* [42], que introduz a utilização na lógica *fuzzy* na rede *ARTMAP*.

Para ilustrar o funcionamento das família de redes *ART*, é possível usar a rede neural *ART 2A*.

A arquitetura da rede *ART 2A* é composta por dois componentes principais: o sistema de atenuação e sistema de orientação, conforme a figura 4.8 [172]. O sistema de atenuação é composto por uma camada de pré-processamento F_0 , uma camada de representação de entradas F_1 e uma camada de representação F_2 . A camada de entrada e representação estão conectados por uma série de pesos adaptativos chamados *bottom-up* ($F_1 \rightarrow F_2$) e *top-down* ($F_2 \rightarrow F_1$).

O caminho do i -ésimo neurônio da camada F_1 para o j -ésimo neurônio da camada F_2 é representado por W_{ij} . Da mesma forma, j -ésimo neurônio da camada F_2 é conectado ao i -ésimo neurônio da camada F_1 pelo peso adaptativo w_{ji} . Esses pesos são usados para multiplicar os sinais enviados entre as camadas e são responsáveis pelo armazenamento do conhecimento obtido pela rede. As interações entre as camadas F_1 e F_2 são controladas pelo sistema de orientação, que emprega o parâmetro de vigilância ρ . O caminho utilizado para atualizar os neurônios é determinado pelo algoritmo de treinamento.

A dinâmica do funcionamento da rede *ART-2A* é determinada pelo parâmetro de vigilância $\rho \in [0, 1]$, e a taxa de aprendizado é controlada pelo parâmetro $\beta \in [0, 1]$. Inicialmente, a camada de saída F_2 não tem nenhuma classe definida, e a camada de entrada é composta pelo conjunto de atributos a serem classificados.

O algoritmo de treinamento da rede *ART-2A* é composto dos seguintes estágios:

- **Pré-processamento:** essa fase normaliza os dados de entrada I^0 :

$$I = \aleph(F_0(\aleph(Vt^0))) \quad (4.30)$$

Na equação 4.30, os parâmetros $\aleph$ e F_0 descrevem as seguintes operações:

$$\aleph(x) \equiv \frac{x}{\|x\|} \equiv \frac{x}{\sum_{i=0}^n x_i^2}, \quad F_0(x) = \begin{cases} x & \text{se } x > \theta \\ 0 & \text{caso contrário} \end{cases} \quad (4.31)$$

Essas operações exercem a normalização Euclidiana e filtragem de ruído sobre os dados de entrada. O processo de remoção de ruído, controlado pelo parâmetro θ , somente fornece resultados adequados quando as principais características do padrão de entrada

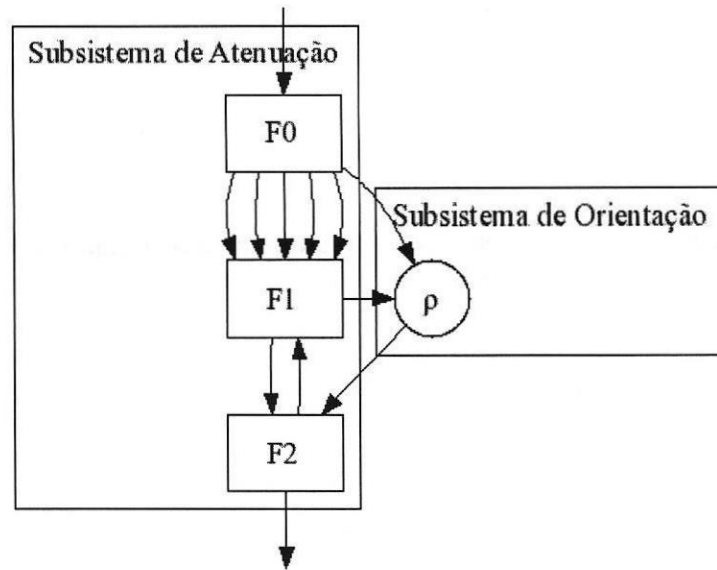


Figura 4.8: Arquitetura da rede ART 2A

(responsáveis pela criação de classes diferentes) estão presentes, exclusivamente, pelos valores mais altos do componente de entrada;

- **Ativação:** essa fase é responsável por enviar os sinais de entrada para os neurônios da camada de representação F_2 , onde T_j corresponde à ativação do neurônio j na camada F_2 .

$$T_j = \begin{cases} Iw_{ij} & \text{se } j \text{ indica um protótipo satisfatório} \\ \alpha \sum_j I_j & \text{caso contrário} \end{cases} \quad (4.32)$$

Inicialmente, todos os neurônios estão marcados como insatisfatórios, e somente passar para satisfatórios quando os seus pesos são adaptados para aprender um determinado padrão de entrada.

O parâmetro de escolha α define o nível máximo da profundidade de busca por um *cluster* adequado. Utilizando o valor $\alpha = 0$, todos os protótipos marcados como satisfatórios são checados antes de escolher um protótipo não-satisfatório como vencedor;

- **Busca:** Essa fase do algoritmo é responsável pela busca do neurônio candidato a armazenar o padrão atual.

O aprendizado competitivo da rede determina o neurônio-candidato em função do número de ativações do neurônio para o padrão atual, conforme a equação 4.33;

$$T_J = \max \{T_j : \text{para todos } F_2 \text{ nodes}\} \quad (4.33)$$

- **Verificação de re-inicialização:** após selecionar o neurônio mais adequado, a condição de re-inicialização é testada:

$$y_J > \rho \quad (4.34)$$

O neurônio candidato é selecionado para armazenar o padrão atual se a equação for verdadeira. Nesse caso, a fase de adaptação é iniciada. Caso contrário, o neurônio vencedor é inibido e a fase de busca é re-inicializada;

- **Adaptação:** essa etapa determina como o padrão será tratado pela rede, que consiste na atualização dos pesos do neurônio vencedor J , que é marcado como satisfatório de acordo com a equação 4.35.

$$w_{Ji}^{new} = \begin{cases} \beta \Psi + (1 - \beta) w_{Ji}^{old} & \text{se } j \text{ representa um protótipo satisfatório} \\ I & \text{caso contrário} \end{cases} \quad (4.35)$$

$$\Psi_i \equiv \begin{cases} I_i & \text{se } w_{Ji}^{old} > \theta \\ 0 & \text{caso contrário} \end{cases} \quad (4.36)$$

A tabela 4.1 ilustra todos os parâmetros utilizados pela rede ART-2A.

Tabela 4.1: Principais parâmetros de ART-2A

Parâmetro	Descrição	Exemplo
m	número de entradas	7
n	número máximo de unidades de representação (<i>clusters</i>)	15
θ	parâmetro de filtragem de ruídos	$\theta = \frac{1}{\sqrt{m}}$
β	taxa de aprendizagem	$\beta = 0.7$
ρ	parâmetro de vigilância	$\rho = 0.9$

O parâmetro de vigilância ρ define o número de classes criadas pela rede. O parâmetro ρ define um intervalo de decisão com o raio de $\sqrt{2(1 - \rho)}$ em torno do vetor de pesos de cada categoria [112]. Para $\rho = 0$, todos os padrões de entrada são agrupados na mesma classe. Tendo $\rho = 1$, é criada uma classe para cada padrão de entrada da rede. A taxa de aprendizado β define a velocidade de adaptação dos protótipos em relação aos padrões de entrada. O uso de $\beta \cong 1$ não é recomendado para a rede ART-2A, uma vez que neste caso os protótipos tendem a migrar entre os padrões associados a uma classe no lugar de convergir para o valor médio dos padrões.

Cada neurônio satisfatório da camada F_2 define um grupo de padrões similares. Dessa forma, os neurônios definem a classificação gerada pela rede para os valores submetidos.

Um dos principais desafios da rede ART-2A é a necessidade de encontrar um valor de vigilância adequado, uma vez que valores altos tendem a criar um número excessivo de *clusters*, e valores baixos não classificam os dados de maneira adequada. Uma das técnicas utilizadas para esse fim consiste no cálculo de distâncias *inter* e *intra-clusters* [172]. Essa abordagem permite determinar um valor ideal do ρ para cada caso de aplicação.

4.3.11 Rede neural SONDE

A rede neural *SONDE* foi proposta por Albertini e Mello [6] para detecção de novidades. Entretanto, ela também pode ser aplicada para classificação de padrões.

A rede é inspirada nas redes neurais *SOM*, *ART* e *GWR*, descritas anteriormente, sendo que ela é utilizada de forma não-supervisionada e *on-line*. O objetivo da rede é a detecção de padrões não-esperados e desconhecidos, resultando em criação de novos neurônios caso tais padrões sejam encontrados, tornando-a adequada para classificação de padrões.

A rede *SONDE* é composta por três camadas: a camada de entrada e pré-processamento, onde os padrões podem ser normalizados caso necessário; a camada de competição, onde os neurônios para cada padrão de entrada são ativados; e a camada de escolha do neurônio mais adequado, onde o neurônio cujo valor de ativação é maior é ajustado, visando representar de melhor forma o padrão da entrada atual.

SONDE visa representar padrões de entrada por meio de neurônios adaptativos. Novos neurônios são criados quando novos padrões são detectados, sendo que cada neurônio c é definido pelo valor de centróide médio dos seus padrões de entrada ω_c , um raio médio de distância entre tais padrões rad_c , e um grau de semelhança α_c , empregado para reconhecer novos padrões.

Os padrões de entrada I_t , sendo t a instante de tempo no qual o padrão é processado, são normalizados por meio da equação 4.37 caso for necessário. Para cada padrão de entrada I_t , o valor de ativação de cada neurônio α_c é calculado através da equação 4.38. Caso nenhum neurônio existente na rede satisfazer a equação 4.39, um novo neurônio é criado, tendo seu centróide ω_{new} inicializado com os valores dos padrões de entrada correspondentes, de forma que $\alpha_{new} = 1$, seu valor de similaridade α_{new} inicializado de acordo com o valor de similaridade pré-definido α_0 e seu raio inicial igual a $rad_{new} = -\ln(\alpha_0)$.

$$I_t = \frac{I_t}{\|I_t\|} \quad (4.37)$$

$$\alpha_c = \exp(-\|I_t - \omega_c\|) \quad (4.38)$$

$$\alpha_t < \alpha_c, \forall c \quad (4.39)$$

Quando a equação 4.39 é satisfeita, o neurônio-vencedor é escolhido por meio da equação 4.40, sendo $\|I - \omega_c\| \leq -\ln \alpha_c$.

$$BMU = \operatorname{argmax}_c(\exp -\|I - \omega_c\|) \quad (4.40)$$

Os parâmetros do neurônio são adaptados através das equações 4.41 e 4.42, onde t representa o intervalo de tempo e γ e ω definem a influência dos padrões anteriores na situação atual.

$$\omega_{BMU_t} = \omega_{BMU_{t-1}} * (1 - \gamma) + I_t * \gamma \quad (4.41)$$

$$rad_{BMU_t} = rad_{BMU_{t-1}} * (1 - \omega) + \|I_t - \omega_{BMU_{t-1}}\| * \omega \quad (4.42)$$

Dessa forma, o centróide e raio do neurônio representa uma média móvel exponencial ponderada dos seus padrões de entrada. Os parâmetros γ e ω influenciam na capacidade de esquecimento da rede, ou seja, quanto maior for o valor deles, mais rápido a rede esquece dos padrões de entrada passados.

Após atualizar o neurônio-vencedor, o seu parâmetro de semelhança α_c é modificado para melhor representar padrões de entrada similares nas próximas ativações do neurônio.

A especialização da rede, isto é, a sua capacidade de detectar padrões de entrada diferentes, é controlada pelo grau de modificação ρ , conforme a equação 4.43.

$$\rho = \left\| \frac{rad_{BMU_t} - rad_{BMU_{t-1}}}{\max(rad_{BMU_t}, rad_{BMU_{t-1}})} \right\| \quad (4.43)$$

Sendo que o valor de ativação do neurônio-vencedor é calculado por meio da equação 4.44.

$$\alpha_{BMU_t} = \min((1 + \rho) * \alpha_{BMU_{t-1}}, \exp(-rad_{BMU_t} * (1 + \rho))) \quad (4.44)$$

Dessa forma, a rede *SONDE* permite uma classificação adaptativa e *on-line* para padrões de entrada heterogêneos, podendo ser aplicada tanto para classificação de padrões quanto para detecção de novidades.

4.3.12 Radial Basis Function

Rede *RBF* (*Radial Basis Function*) é uma rede neural amplamente utilizada para classificação de padrões e previsões de séries temporais [28, 27, 234, 63, 26]. A rede visa calcular uma função de regressão, que representa os padrões detectados através de um conjunto de funções gaussianas, conforme a figura 4.9, onde a função resultante é calculada a partir de 5 funções auxiliares.

A rede *RBF* é composta por três camadas: uma camada de entrada, uma escondida com função não-linear de ativação de *RBF*, e uma de saída, conforme apresentado na figura 4.10.

O valor de saída da rede é calculado através da equação 4.45, sendo N o número de neurônios na camada escondida, c_i o valor de centróide para neurônio i , e α_i representando os pesos do neurônio de saída. O valor das funções gaussianas ρ é calculado por meio da equação 4.46, sendo que β é uma constante de peso pré-determinada.

$$\phi(x) = \sum_{i=1}^N \alpha_i \rho(\|x - c_i\|) \quad (4.45)$$

$$\rho(\|x - c_i\|) = \exp(-\beta \|x - c_i\|^2) \quad (4.46)$$

A rede *RBF* é aplicada em diversas áreas, sendo utilizada para classificação de padrões, predição de séries temporais e detecção de novidades [28, 27, 234, 63, 26].

4.3.13 Redes neurais separadoras

As técnicas de classificação e clusterização, apresentadas anteriormente, visam agrupar padrões de entrada em diversos *clusters*, em função da semelhança entre eles. Entretanto, tais redes não permitem fazer o processo inverso, ou seja, extrair padrões individuais a partir de uma entrada composta.

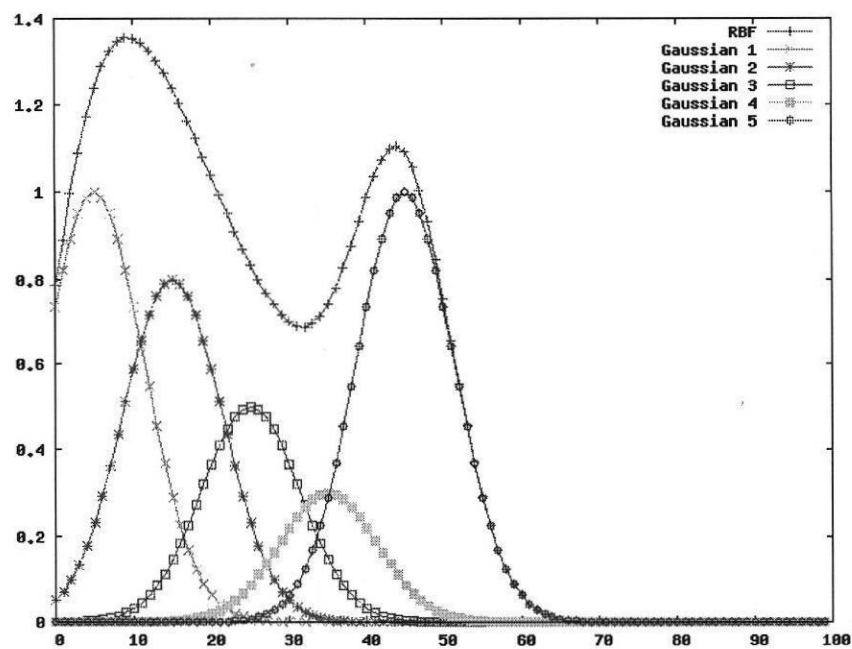


Figura 4.9: Rede neural *RBF*

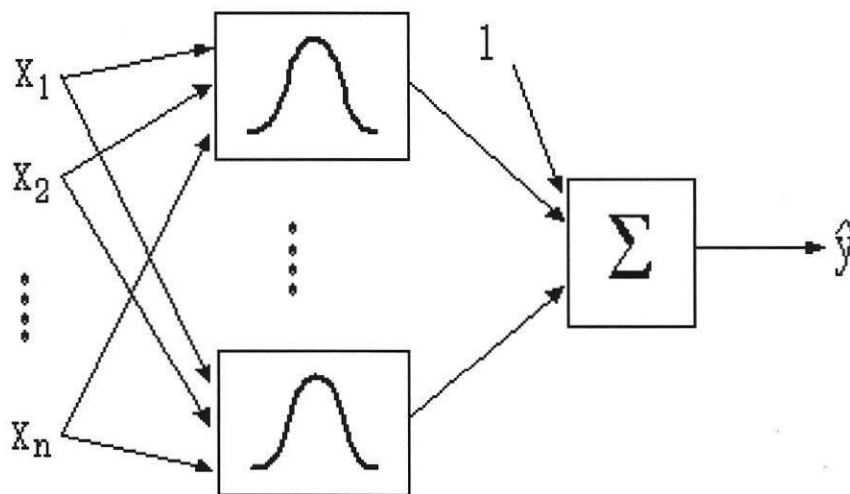


Figura 4.10: Arquitetura da rede *RBF*

Para solucionar esse problema, é possível empregar técnicas de separação e classificação de sinais independentes, extraindo uma série de sinais aditivos a partir de um sinal composto, conforme a figura 4.11 [225].

Supondo que o sinal composto apresentado na figura 4.11 representa um sinal composto, é possível determinar e classificar os processos independentes de maneira automática através de um método conhecido como *Blind Signal Separation (BSS)*. Como a aplicação desse método

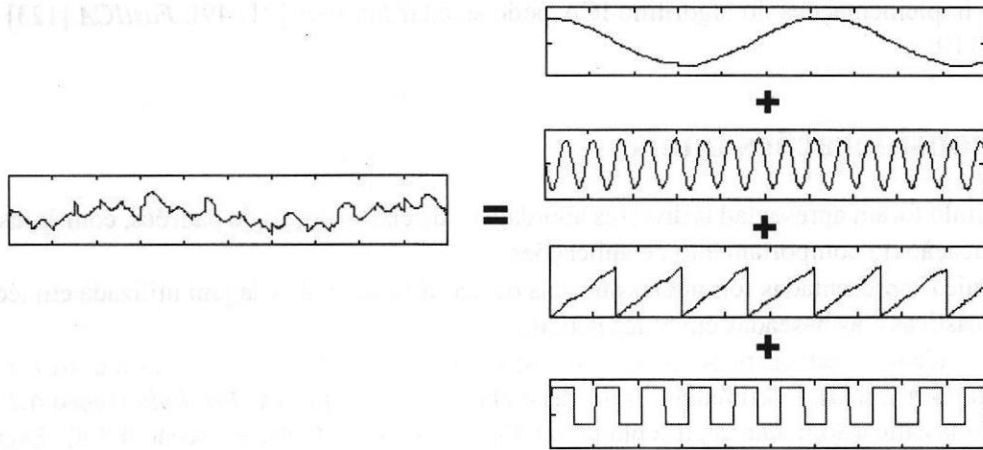


Figura 4.11: ICA: decomposição de sinais

pode-se citar redes neurais ICA [189, 109, 80].

As redes neurais ICA (*Independent Component Analysis*) são utilizadas para a separação de um sinal composto por diversos fatores ou fontes em seus sub-componentes aditivos. Supondo uma independência estatística mútua entre os sinais originais, a sua extração do sinal composto é possível por meio de maximização da independência estatística entre os componentes estimados, que representam os sinais individuais.

Representando os dados de um sinal composto por meio de um vetor $x = (x_1, \dots, x_m)^T$, é possível representá-lo por meio de uma soma de componentes independentes s_k , onde $k = 1, \dots, n$, conforme a equação 4.47.

$$x_i = a_{i,1}s_1 + \dots + a_{i,k}s_k + \dots + a_{i,n}s_n \quad (4.47)$$

Dessa forma, o modelo pode ser escrito em forma vetorial conforme a equação 4.48, onde o vetor observado x é representado por vetores-base $a_k = (a_{1,k}, \dots, a_{m,k})^T$.

$$x = \sum_{k=1}^n a_k s_k \quad (4.48)$$

Representando os vetores-base a_k como as colunas de matriz de composição $A = (a_1, \dots, a_n)$, pode-se escrever a formula de composição de sinais como $x = As$, sendo $s = (s_1, \dots, s_n)^T$.

Tendo os dados do vetor composto $x (x_1, \dots, x_N)$, a tarefa consiste em determinar os sinais independentes $s (s_1, \dots, s_n)$ e a matriz de composição A . Dessa forma, para determinar os sinais independentes é suficiente multiplicar o vetor x pela matriz A inversa, obtida da equação 4.49.

$$W = A^{-1} \quad (4.49)$$

A matriz resultante W é conhecida como a matriz de decomposição de dimensão $n * n$, utilizada para calcular os n sinais independentes s a partir do sinal composto x .

Como aplicações de ICA é possível mencionar o reconhecimento de voz, separação de múltiplas fontes de dados, processamento de imagens e aplicações biomédicas [189]. Como

principais implementações do algoritmo ICA pode-se citar *infomax* [21, 49], *FastICA* [123] e *MF-ICA* [119].

4.4 Considerações finais

Neste capítulo foram apresentadas diversas abordagens de classificação de padrões, com ênfase em classificação do comportamento de aplicações.

As técnicas apresentadas foram classificadas de acordo com a abordagem utilizada em técnicas estocásticas e as baseadas em redes neurais.

Entre as técnicas estocásticas foram abordados os modelos de clusterização iterativa, representado por técnicas de *k*-médias, *c*-médias e abordagem de *quality threshold* (seção 4.2 e técnica de classificação linear, representada por *Support Vector Machines* (seção 4.2.4). Estas abordagens possibilitam realizar a classificação de acordo com as diferenças observadas, agrupando padrões similares em classes independentes de acordo com os parâmetros de classificação pré-definidos.

Entre as técnicas baseadas em redes neurais, foi apresentada a técnica de agrupamento *SOM* na seção 4.3.1, junto com diversas abordagens auto-expansíveis (seção 4.3.2) e redes voltadas para detecção de novidades, como *GWR* (seção 4.3.9) e *SONDE* 4.3.11. Também foram apresentadas abordagens para classificação de acordo com a ressonância entre os padrões, com a rede *ART* (seção 4.3.10); abordagem baseada em representação de padrões por meio de funções radiais (rede *RBF*, apresentada na seção 4.3.12).

Além disso, uma abordagem de classificação para classificação de padrões independentes embutidos num sinal composto foi abordada, conhecida como *Blind Signal Separation*, sendo representada por rede *ICA* (seção 4.3.13)

Cada uma das técnicas abordadas apresenta vantagens para casos específicos, sendo que nenhuma dela é adequada para ser utilizada como uma solução única para todos os casos.

Capítulo 5

Predição do comportamento de processos

5.1 Considerações iniciais

A predição do comportamento de processos é essencial para o entendimento do funcionamento de um sistema distribuindo, possibilitando a aplicação do conhecimento predito a partir das observações passadas para melhorias nas técnicas de leitura antecipada de dados, acesso à memória distribuída e balanceamento de carga e escalonamento distribuídos [72, 65, 99, 209, 9, 205, 202, 200, 171].

Entre as diversas técnicas estatísticas utilizadas para a predição, é possível mencionar as baseadas no estado atual do processo, tais como cadeias de Markov e modelo escondido de Markov com suas extensões, apresentadas na seção 5.2; modelos que representam o comportamento por meio de uma equação é utilizada pelas técnicas auto-regressivas (seção 5.3); e a técnica de filtro de Kalman, que visa realizar predições de tendências do comportamento em sistemas com ruídos, descrito na seção 5.4.

Diversas redes neurais voltadas para predição de comportamento são apresentadas nas seções 5.5, apresentando as redes recorrentes e rede neural *LSTM*, rede de Hopfield e máquinas de Boltzmann; 5.6, descrevendo diversas redes neurais que utilizam aprendizado competitivo para realizar predições; e 5.7, descrevendo a família de redes neurais baseadas em atrasos no tempo.

Técnicas baseadas no aprendizado Bayesiano, por sua vez, são apresentadas na seção 5.10, e conceitos de teoria de informação e entropia são introduzidos na seção 5.11. Abordagens utilizadas nas técnicas conhecidas como *Reservoir Computing* são apresentadas na seção 5.8, e seção 5.12 descreve os conceitos de algoritmos genéticos e computação evolutiva. Finalmente, a seção 5.13 sumariza as técnicas abordadas e conclui este capítulo.

5.2 Técnicas baseadas no estado atual

As técnicas de predição baseadas no estado atual do processo são caracterizadas por avaliar somente o atual momento de execução, sem considerar o histórico de comportamentos. Dessa forma, assume-se que existe somente um fluxo de execução através do qual é possível chegar no estado de execução corrente.

Tabela 5.1: Exemplo de matriz de incidência

Estado original	Novo estado	Probabilidade
<i>Execução</i>	<i>I/O</i>	30%
<i>Execução</i>	<i>Espera</i>	20%
<i>Execução</i>	<i>Execução</i>	50%
<i>I/O</i>	<i>Espera</i>	30%
<i>I/O</i>	<i>I/O</i>	70%
<i>Espera</i>	<i>I/O</i>	40%
<i>Espera</i>	<i>Execução</i>	60%

5.2.1 Cadeias de Markov

Entre diversos modelos analíticos utilizados para a determinação e predição de comportamento de aplicações pode-se citar Cadeias de Markov [27].

Uma cadeia de Markov de tempo discreto é um processo estocástico que apresenta a propriedade de Markov [27], demonstrada na equação 5.1, no qual a predição de um próximo estado é feita considerando somente o estado atual de execução do processo.

$$Pr(X_{n+1} = x | X_0, X_1, X_2, \dots) = Pr(X_{n+1} = x | X_n) \quad (5.1)$$

Uma cadeia de Markov é definida por uma sequência $X_1, X_2, X_3, \dots$ de variáveis aleatórias, sendo que o seu escopo é denominado de espaço de estados e X_n denota o estado do processo na instante n . Caso a distribuição de probabilidade condicional X_{n+1} for determinada somente em função do seu valor no estado anterior X_n , têm-se a equação 5.1 satisfeita, onde x representa o estado atual do sistema e $x + 1$ o próximo estado.

Uma das técnicas de visualização de uma cadeia de Markov consiste em sua representação por meio de uma máquina de estados finitos. Se o processo encontra-se no estado y no tempo n , a probabilidade da sua transição para o estado x no tempo $n + 1$ é independente do tempo n , sendo determinada somente em função do seu estado atual y . Dessa forma, em qualquer instante n , uma cadeia de Markov finita pode ser caracterizada por uma matriz de incidência cujo elemento (x, y) é independente do tempo n , e é calculado pela equação 5.2.

$$Pr(X_{n+1} = x | X_n = y) \quad (5.2)$$

Cadeias de Markov também podem ser descritas por meio de grafos dirigidos, cujos nós representam os estados de execução e as arestas são rotuladas com a probabilidade de transição entre eles.

As cadeias de Markov podem ser aplicadas para prever o comportamento de aplicações, determinando seus estados de execução e probabilidades de transições entre eles. Por exemplo, considerando uma aplicação produtor-consumidor no estado de *execução*, com probabilidades de 30% de transição para o estado de *transmissão de dados*, de 20% para o estado de *espera* e de 50% de permanecer no estado atual, a matriz de incidência completa desse caso pode ser vista na tabela 5.1 e a cadeia de Markov correspondente, na figura 5.1. Com esses valores, pode-se prever as ações a serem tomadas, baseando-se somente no estado atual de execução e desconsiderando o histórico de comportamentos.

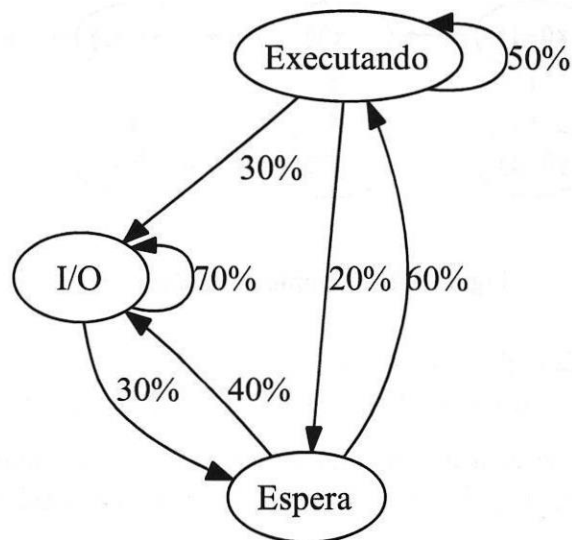


Figura 5.1: Exemplo de cadeia de Markov

Diversas extensões e aplicações ao modelo de cadeias de Markov foram propostas, sendo apresentadas a seguir.

5.2.2 Modelo Markov escondido

O Modelo Markov Escondido (*Hidden Markov Model*, ou *HMM*) é um modelo estatístico no qual o sistema é visto como um processo de Markov com parâmetros desconhecidos, representando os estados de execução do processo; e parâmetros observáveis, visíveis para o observador. O objetivo deste modelo é determinar os parâmetros desconhecidos a partir dos observáveis [173, 174, 188, 191].

Diferentemente do modelo de Markov tradicional, onde o estado é visível ao observador e as probabilidades de transições entre os estados são os únicos parâmetros do sistema, no *HMM* o estado não é diretamente acessado, ao contrário das variáveis influenciadas por ele. A distribuição de probabilidade de cada estado é calculada indiretamente avaliando a sequência de símbolos de observação gerados pelo sistema.

A arquitetura de *HMM* é composta por uma cadeia Markov escondida e não observável, com um número finito de estados, um estado de transição, matriz de probabilidade de transições e um estado inicial; e um conjunto de funções de probabilidade associadas a cada estado. Formalmente, o modelo é composto por seguintes elementos:

1. Um conjunto de estados $S = S_1, S_2, \dots, S_N$, sendo N o número de estados no modelo. O estado do modelo no tempo t é dado por $q_t \in S$, sendo que $1 \leq t \leq T$, sendo T o tamanho da sequência de observação;
2. O conjunto de diferentes símbolos de observação $V = v_1, v_2, \dots, v_M$, sendo M o número de diferentes símbolos no modelo;

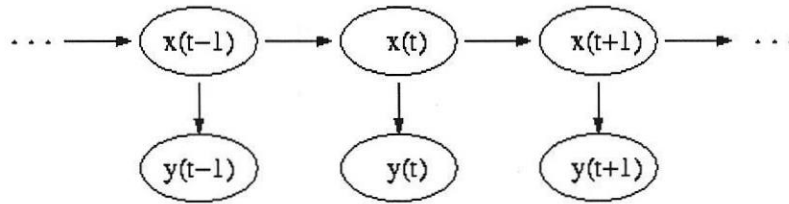


Figura 5.2: Exemplo de HMM

3. Matriz de probabilidade de transição entre os estados $A = \alpha_{ij}$, sendo $\alpha_{ij} = P[q_t = S_j | q_{t-1} = S_i]$, com $1 \leq i, j \leq N$, $0 \leq \alpha_{i,j} \leq 1$, $\sum_{j=1}^N \alpha_{ij} = 1$ e $1 \leq i \leq N$;
4. Símbolo de observação da matriz de probabilidade $B = b_j(k)$, onde $b_j(k) = \rho[O_t = v_k | q_t = S_j]$, com $1 \leq j \leq N$ e $1 \leq k \leq M$. O símbolo de observação no tempo t é representado por O_t ;
5. Uma distribuição de estado π inicial, $\pi = \pi_i$, sendo que $\pi_i = P[q_1 = S_i]$ com $1 \leq i \leq N$.

Dessa forma, um HMM λ pode ser formalmente representado pela equação 5.3.

$$\lambda = (A, B, \pi) \quad (5.3)$$

Um exemplo de HMM é apresentado na figura 5.2, sendo que $x(t)$ representa o estado escondido no intervalo de observação t , e $y(t)$ é a observação ocorrida no mesmo intervalo, sendo que as flechas representam as dependências entre as transições de estados. Dessa forma, é possível ver que a transição do estado $x(t)$ para $x(t+1)$ depende somente do valor de $x(t)$. Da mesma forma, os valores da variável observada $y(t)$ no instante t dependem somente do valor resultante no estado $x(t)$. Avaliando a sequência de observações $y(1, \dots, N)$ é possível determinar a sequência de estados de execução correspondentes, e as probabilidade de transição entre eles.

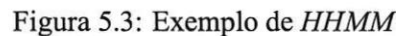
O modelo HMM é amplamente utilizado para predição do comportamento de processos. Diferentemente do modelo de Markov tradicional, que permite predizer somente uma transição de estado, o HMM possibilita realizar predição de sequências de transições de estados, conforme a equação 5.4, sendo que a probabilidade de observar sequência $Y = y(0), y(1), \dots, y(L-1)$ de tamanho L é representada pelo valor de $P(Y)$, considerando que $X = x(0), x(1), \dots, x(L-1)$ representa todas as possíveis sequências de transições entre os estados escondidos.

$$P(Y) = \sum_X P(Y|X)P(X) \quad (5.4)$$

5.2.3 Modelo HMM Hierárquico

O modelo HMM Hierárquico (HHMM) é uma extensão do HMM, que considera que estado de execução como sendo um modelo probabilístico auto-contido [86]. A arquitetura do modelo pode ser vista na figura 5.3.

A principal vantagem deste modelo em relação ao HMM é o fato de que cada estado do HHMM é composto por um modelo independente. Desta forma, os estados podem emitir séries de símbolos para cada transição, ao invés de símbolos independentes como acontece com



5.2.4 Modelo *HMM* em níveis

A principal vantagem desse modelo é a possibilidade de realizar uma generalização mais controlada, uma vez que cada nível do *LHMM* pode empregar uma granularidade diferente.

5.3 Técnicas auto-regressivas

A representação do comportamento de uma aplicação por meio de uma série temporal, ou um modelo auto-regressivo, permite realizar predições por meio de técnicas estatísticas. Entre alguns dos métodos mais conhecidos é possível mencionar os modelos auto-regressivos não-lineares, para detecção de padrões mais complexos; e a técnica de auto-correlação para detectar periodicidades em uma série temporal.

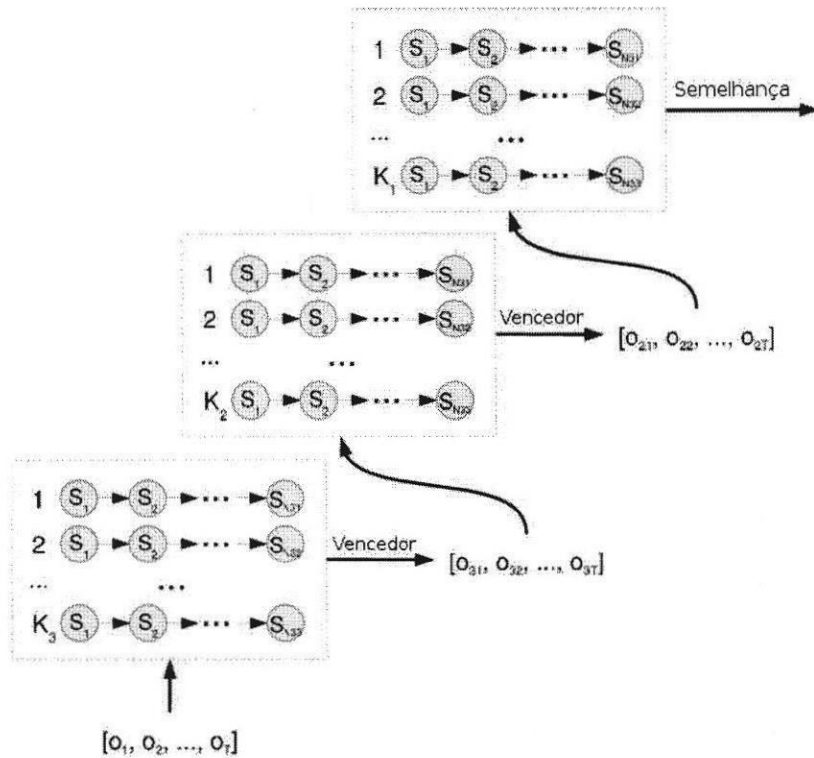


Figura 5.4: Exemplo de *LHMM*

5.3.1 Auto-regressão linear

A técnica conhecida como auto-regressão linear, ou simplesmente regressão, é uma das técnicas mais utilizadas para representação do comportamento [237, 125], e visa descrevê-lo por meio de um modelo matemático, conforme equação 5.5.

$$y(k) = \sum_{n=1}^T a(n)y(k-n) + e(k) = \hat{y}(k) + e(k) \quad (5.5)$$

O modelo representa $y(k)$ como a soma ponderada dos valores dos elementos anteriores na sequência, sendo que a predição do próximo elemento da sequência é calculada por $\hat{y}(k)$. O erro da predição é determinado através de $e(k) = y(k) - \hat{y}(k)$, e é geralmente considerado como ruído.

Modelos mais avançados estendem a técnica de regressão para *não-linear*, onde N é o resultado de uma rede neural calculado em função de T valores $y(k-1), \dots, y(k-T)$ prévios que representam os estados anteriores, conforme a equação 5.6.

$$y(k) = \hat{y}(k) + e(k) = N[y(k-1), y(k-2), \dots, y(k-T)] + e(k) \quad (5.6)$$

5.3.2 Modelos não-lineares

Enquanto o modelo auto-regressivo possibilita definir o comportamento de um processo de forma linear, frequentemente isso não representa precisamente o comportamento dele. Para isso, algumas técnicas foram propostas, como o modelo *SVCA*.

A abordagem utilizada pelo modelo *Sinusoidally Varying Coefficients Autoregressive Model (SVCA)* [182] emprega técnica de auto-regressão não-linear para construir uma representação do comportamento do processo. O modelo utiliza a equação 5.7 para prever o próximo estado de execução do processo y_t , sendo u_i calculado por meio da equação 5.8, N o número de estados anteriores avaliados, ϵ_t uma variável aleatória com média 0 e variância $\chi^2 \forall t$, e função γ definida pela equação 5.9, onde os coeficientes a e b da equação 5.9 determinam o quanto o valor atual de u_t é influenciado pelas observações anteriores.

$$y_t = y_{t-1} + u_i \quad (5.7)$$

$$u_i = \gamma_0(t) + \sum_{r=1}^N \gamma_r(t) u_{t-r} + \epsilon_t \quad (5.8)$$

$$\gamma_r(t) = a_{t,r} \sin\left(\pi \frac{t - a_{2,r}}{a_{3,r}}\right) + b_{t,r} \cos\left(\pi \frac{t - b_{2,r}}{b_{3,r}}\right) \quad (5.9)$$

Outros modelos não-lineares de auto-regressão também são utilizados frequentemente, como o modelos *Non-Linear Auto-Regressive Exogeneous Model (NARX)* [30] e *Auto-Regressive Moving Average Model with Exogeneous Inputs (ARMAX)* [30], voltados para predição de séries com variáveis independentes.

O modelo *NARX* é definido pela equação 5.10, onde y_t é o estado predito na instante de observação t , u uma variável auxiliar e ϵ_t o erro residual. Nesse caso, uma vez que as variáveis da série são independentes, as informações contidas em u são utilizadas na predição do próximo estado da sequência. Por exemplo, y pode representar a temperatura média diária, e u o dia de ano. A função F geralmente é uma função polinomial ou uma rede neural.

$$y_t = F(y_{t-1}, y_{t-2}, y_{t-3}, \dots, u_{t-1}, u_{t-2}, u_{t-3}, \dots) + \epsilon_t \quad (5.10)$$

O modelo *ARMAX*, por sua vez, pode ser descrito formalmente como sendo *ARMAX(p, q, b)*, definindo o modelo com p termos auto-regressivos, q termos que representam a média móvel e b entradas independentes (ou *exogêneas*). Dessa forma, a predição do estado X_t no intervalo de observação t é realizada pela equação 5.11, sendo $\eta_1, \dots, \eta_b$ os parâmetros independentes da entrada d_t , $\phi_1, \dots, \phi_p$ os parâmetros do modelo auto-regressivo e $\theta_1, \dots, \theta_q$ os valores da média móvel de erro.

$$X_t = \epsilon_t + \sum_{i=1}^p \phi_i X_{t-i} + \sum_{i=1}^q \theta_i \epsilon_{t-i} + \sum_{i=1}^b \eta_i d_{t-i} \quad (5.11)$$

Os modelos de auto-regressão não-linear são particularmente eficazes em casos onde os padrões de entrada não apresentam nenhuma co-relação facilmente observada.

5.3.3 Auto-correlação

Técnicas de auto-correlação visam detectar padrões repetitivos em sinais através de correlação do processo com ele mesmo, porém deslocado no tempo [128]. Dessa forma, considerando o processo X_t um processo estocástico discreto com média μ , a função de auto-correlação pode ser descrita conforme a equação 5.12, onde E representa o valor esperado, k é o deslocamento no tempo (também conhecido como *lag*) e σ^2 é uma constante de normalização.

$$R(k) = \frac{E[(X_i - \mu)(X_{i+k} - \mu)]}{\sigma^2} \quad (5.12)$$

Dessa maneira, o domínio da função encontra-se no intervalo $[-1, 1]$, sendo que o valor de 1 indica uma correlação perfeita e -1 significa a existência de uma anti-correlação perfeita.

Da mesma forma, uma auto-correlação contínua de um sinal $f(x)$ pode ser encontrada por meio de uma correlação contínua do $f(x)$ com ele mesmo, após um deslocamento de τ , conforme a equação 5.13, onde f^* representa o conjugado complexo e $\circ$ representa a convolução entre as duas funções.

$$R_f(\sigma) = f^*(-\sigma) \circ f(\sigma) = \int_{-\inf}^{\inf} f(t + \sigma) f^*(t) dt = \int_{-\inf}^{\inf} f(t) f^*(t - \sigma) dt \quad (5.13)$$

Formalmente, uma auto-correlação discreta R com *lag* j para um sinal x_n é dada pela equação 5.14, onde m é o valor esperado de x_n .

$$R(j) = \sum_n (x_n - m)(x_{n-j} - m) \quad (5.14)$$

Uma vez que os sinais podem ser compostos por vetores multi-dimensionais, a função de auto-correlação pode ser adaptada de maneira correspondente. Por exemplo, para um sinal tridimensional, a auto-correlação é definida pela equação 5.15.

$$R(j, k, l) = \sum_{n, q, r} (x_{n, q, r} - m)(x_{n-j, q-k, r-l} - m) \quad (5.15)$$

Além disso, a técnica de auto-correlação pode ser aplicada para determinar o intervalo no qual as mudanças de comportamento aconteceram, e ajustar o mecanismo de predição adequadamente [72].

5.4 Filtro de Kalman

O filtro de Kalman [131, 67, 186, 230, 23, 142, 180] é uma técnica de predição que atua sobre sistemas dinâmicos discretos ao longo do tempo determinando o estado verdadeiro do processo num sistema com ruídos.

O estado atual do sistema é representado por um vetor de números reais, sendo que, a cada incremento do tempo, um operador linear é aplicado para determinar o próximo estado do sistema.

O modelo é bastante similar ao modelo *HMM* [173, 174, 188, 191], com a diferença de que, enquanto este opera sobre variáveis discretas, o filtro de Kalman utiliza variáveis contínuas.

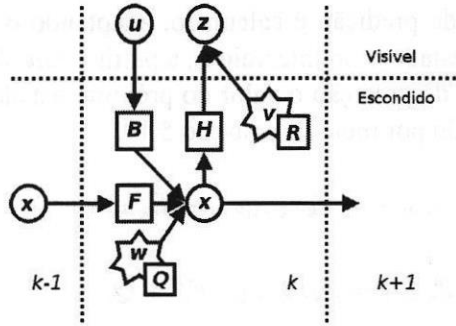


Figura 5.5: Exemplo de filtro de Kalman

Ao utilizar o filtro de Kalman para avaliar o estado interno do processo a partir de uma sequência de amostras com ruído presente, é necessário modelar o processo de acordo com o *framework* de Kalman, apresentado na figura 5.5, sendo que círculos representam os vetores; quadrados as matrizes, e estrelas representam ruído gaussiano com a matriz de co-variância associada. Desta forma, para cada intervalo de observação k , é necessária a definição de matrizes F_k , H_k , Q_k e, em alguns casos, B_k , conforme descrito a seguir.

O filtro de Kalman assume que o estado verdadeiro do processo $x(k)$ no intervalo k é uma evolução a partir do seu estado no intervalo $k - 1$ de acordo com a equação 5.16, sendo que:

$$x_k = F_k x_{k-1} + B_k u_{k-1} + w_{k-1} \quad (5.16)$$

1. F_k é o modelo de transição de estados, aplicado ao estado anterior x_{k-1} ;
2. B_k é o modelo de controle de entrada, aplicado ao vetor de controle u_k ;
3. w_k representa o ruído do processo, sendo uma distribuição normal multi-variada com co-variância Q_k , definida por $w_k \sim N(0, Q_k)$.

Num intervalo de observação k , o estado x_k é representado por z_k , conforme a equação 5.17, sendo que H_k define o modelo de observação que mapeia o estado verdadeiro do processo no espaço de observação, e v_k é o ruído de observação, que é considerado um ruído branco gaussiano com co-variância R_k , sendo que $v_k \sim N(0, R_k)$.

$$z_k = H_k x_k + v_k \quad (5.17)$$

No modelo, o estado inicial e os vetores de ruído $x_0, w_1, \dots, w_k, v_1, \dots, v_k$ são mutuamente independentes.

O funcionamento das técnicas de predição baseadas no filtro de Kalman é recursivo, ou seja, para predizer um próximo estado a partir do atual, somente a sequência de estados imediatamente anteriores é avaliada, desconsiderando o histórico de comportamento passados. O estado do filtro no intervalo de observação k é representado por duas variáveis: $\hat{x}_{k|k}$, sendo o valor estimado do estado; e $P_{k|k}$, que corresponde à matriz de co-variância de erro, utilizada para avaliar a precisão da predição.

O filtro emprega duas fases de funcionamento: a de predição, e a de atualização. Durante a fase de predição, as informações do estado atual são utilizadas para predizer o próximo.

Na fase de atualização, o erro de predição é calculado, adaptando o filtro. A notação $\hat{x}_{n|m}$ representa o valor estimado do estado x no intervalo n , a partir de m observações passadas.

Dessa forma, durante a fase de predição o valor do próximo estado é estimado a partir da equação 5.18, e o erro é calculado por meio da equação 5.19.

$$\hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1} + B_k u_k \quad (5.18)$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \quad (5.19)$$

Na fase de atualização, o modelo é adaptado às novidades observadas no sistema, possibilitando o cálculo de:

1. Inovação, ou resíduo da amostra:

$$\hat{y}_k = z_k - H_k \hat{x}_{k|k-1};$$

2. Co-variância residual:

$$S_k = H_k P_{k|k-1} H_k^T + R_k;$$

3. Ganho ótimo:

$$K_k = P_{k|k-1} H_k^T S_k^{-1};$$

4. Próximo estado estimado atualizado:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \hat{y}_k;$$

5. Co-variância estimada atualizada:

$$P_{k|k} = (I - K_k H_k) P_{k|k-1}.$$

O filtro de Kalman é amplamente utilizado para predição de tendências do comportamento em ambientes com ruídos ou perdas de dados.

5.4.1 Modelo de Kalman suavizado: *Kalman Smoother*

A técnica de predição denominada de *Kalman Smoother* [52, 79, 194, 77] estende o filtro de Kalman, suavizando a distribuição das distribuições de estados recursivamente.

Para isso, a equação 5.20 é aplicada sobre as equações do filtro de Kalman 5.18 e 5.19.

$$\rho(x_k | y_{1:T}) = N(x_k | m_k^S, P_k^S) \quad (5.20)$$

Desta forma, a distribuição de resultados é suavizada, confinando os resultados ao plano $y_{1:T}$, enquanto os resultados originais são calculados somente em relação aos resultados obtidos até o intervalo de amostragem k , ou seja, pertencentes a $y_{1:k}$. Tal distribuição pode ser calculada a partir dos resultados do filtro de Kalman, aplicando recursivamente as equações 5.21, 5.22, 5.23, 5.24, iniciando-se no intervalo amostral T , com $m_T^S = m_T$ e $P_T^S = P_T$.

$$P_{k+1}^- = A_k P_k A_k^T + Q_k \quad (5.21)$$

$$C_k = P_k A_k^T [P_{k+1}^-]^{-1} \quad (5.22)$$

$$m_k^S = m_k + C_k[m_{k+1}^S - A_k m_k] \quad (5.23)$$

$$P_k^S = P_k + C_k[P_{k+1}^S - P_{k+1}]C_k^T \quad (5.24)$$

Para melhorar a eficiência da predição de curto alcance, a técnica introduz um modelo auto-regressivo com pesos variáveis de acordo com o modelo aleatório *Guassiano*, conforme pode ser visto nas equações 5.25 e 5.26, sendo que o ruído do processo v_k^{ar} tem média de 0 e co-variância $Q = q^{ar}I$. O vetor de pesos w_k é calculado a partir das amostras conhecidas, e o ruído de amostragem tem distribuição *Gaussiana* $r_k^{ar} \sim N(0, \sigma_{ar}^2)$.

$$w_k = w_{k-1} + v_k^{ar} \quad (5.25)$$

$$e_k = \sum_i w_{i,k} e_{k-1} + r_k^{ar} \quad (5.26)$$

Tendo estimado o modelo de auto-regressão a partir dos dados originais, a estimação final é obtida das equações 5.27 e 5.28, sendo $r_k^P \in N(0, \sigma^2)$, e o ruído de processo v_k^P tem a variância de q^P .

$$d_k = \sum_i w_{i,j} d_{k-1} + v_k^P \quad (5.27)$$

$$e_k = d_k + r_k^P \quad (5.28)$$

A estimativa final do sinal é calculada como sendo $\hat{y}_k = \hat{x}_k + \hat{d}_k$, sendo $\hat{x}$ o resultado de predição realizado pelo filtro de Kalman suavizado, e $\hat{d}^k$ calculado pelo modelo de predição de curto alcance.

Dessa forma, duas abordagens diferentes são aplicadas para realizar predições de curto e longo alcance:

1. A previsão de longo alcance é realizada em duas etapas:
 - (a) O filtro de Kalman é aplicado sobre a seqüência de dados de entrada, armazenando os valores da predição realizada e da co-variância. As amostras faltantes são preditas;
 - (b) O suavizador é executado sobre os resultados estimados no passo anterior, resultando em uma nova série temporal com os valores preditos.
2. A previsão de curto alcance, por sua vez, consiste de quatro etapas:
 - (a) O filtro de Kalman é executado sobre os resultados da predição realizada pelo modelo de predição de curto alcance, apresentado nas equações 5.25 e 5.26, determinando os vetores auto-regressivos de pesos. Os pesos das amostras faltantes são preditos;
 - (b) O suavizador é aplicado sobre os resultados estimados no passo anterior, resultando em uma série temporal suavizada com os valores preditos;

- (c) O filtro de Kalman é aplicado sobre a seqüência residual, calculada nas equações 5.27 e 5.28, determinando a periodicidades nos padrões de curto alcance. Tal periodicidade também é predita para as amostras faltantes;
- (d) O suavizador é aplicado sobre os resultados estimados no passo anterior, resultando em uma série temporal suavizada, contendo os valores preditos.

Dessa forma, o modelo permite predição eficiente tanto de curto quanto de longo alcance [77].

5.4.2 Filtro de Kalman estendido

Filtro estendido de Kalman é uma evolução do modelo de Kalman, que visa a utilização de funções derivadas no lugar de lineares, como no modelo de Kalman original [217].

As equações 5.29 e 5.30 são utilizadas para o cálculo de predições de estados, sendo que a função f é utilizada para prever próximo estado a partir do último estado predito; e função h retorna o estado previsto na etapa anterior a partir do estado atual.

$$x_k = f(x_{k-1}, u_k, w_l) \quad (5.29)$$

$$z_k = h(h_k, v_k) \quad (5.30)$$

No lugar de aplicar as funções f e h diretamente na co-variância, uma matriz Jacobiana de derivadas parciais é calculada, sendo re-avaliada para cada nova amostra. Introduzindo a matriz no filtro de Kalman original, as seguintes equações são obtidas:

- Fase de predição:

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k}, u_k, 0) \quad (5.31)$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \quad (5.32)$$

- Fase de atualização:

1. Inovação, ou resíduo da amostra:

$$\hat{y}_k = z_k - h(\hat{x}_{k|k-1}, 0)$$

2. Co-variância residual:

$$S_k = H_k P_{k|k-1} H_k^T + R_k;$$

3. Ganho ótimo:

$$K_k = P_{k|k-1} H_k^T S_k^{-1};$$

4. Próximo estado estimado atualizado:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \hat{y}_k;$$

5. Co-variância estimada atualizada:

$$P_{k|k} = (I - K_k H_k) P_{k|k-1}.$$

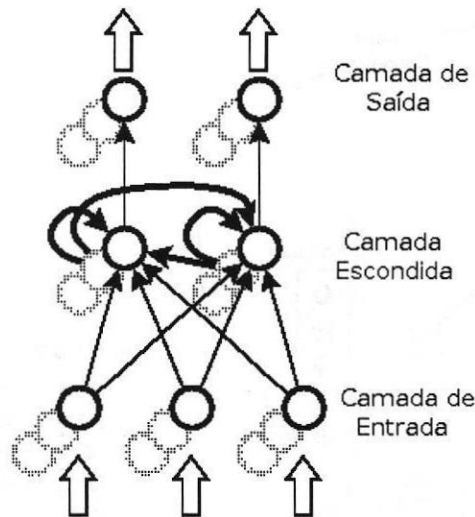


Figura 5.6: Exemplo de uma rede neural recorrente

Sendo que as matrizes de transição e observação são definidas por meio das equações 5.33 e 5.34.

$$F_k = \frac{\partial f}{\partial h} \bigg|_{\hat{x}_{k|k-1}, u, k} \quad (5.33)$$

$$H_k = \frac{\partial h}{\partial f} \bigg|_{\hat{x}_{k|k-1}} \quad (5.34)$$

Como um dos criticismos do modelo estendido do filtro de Kalman é possível mencionar o fato de que o modelo não resulta em um resultado ótimo, podendo se limitar a um mínimo local. Além disso, caso a predição inicial não for precisa, o filtro pode se divergir rapidamente. O modelo também prediz as tendências a partir dos resultados mais recentes, o que nem sempre representa o comportamento corretamente.

Atualmente, o modelo estendido do filtro de Kalman é bastante utilizado em casos onde é necessário realizar predições sobre tendências no meio de ruídos, tais como sistemas de navegação e GPS [137, 190, 66].

Entre outras extensões do modelo é possível citar *Weighted Bidirectional Multi-stream Extended Kalman Filter*, que utiliza técnicas de *back-propagation* e separação de dados de entrada em múltiplos fluxos distintos [121].

5.5 Redes neurais recorrentes

Redes recorrentes compõem uma família de redes neurais que apresentam conexões entre as unidades em forma de um grafo cíclico direcionado, conforme a figura 5.6 [153, 78, 216, 162, 239].

As redes neurais recorrentes são caracterizadas pela utilização de conexões para armazenamento de eventos recentes, em forma de memória de curto prazo (*short-term memory*).

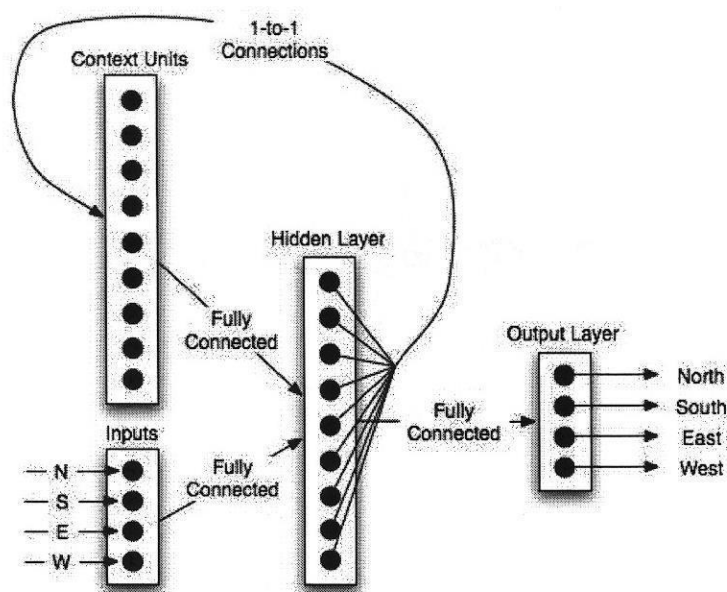


Figura 5.7: Rede neural de Elman

Como exemplo de rede neural recorrente, cita-se a rede de Elman [153, 216], caracterizada por introduzir uma camada especial de cópia, denominada de contexto, conectada à camada escondida e utilizada para manter o conhecimento sobre o estado anterior da rede. Diversas camadas de cópia podem ser utilizadas, possibilitando representar um número arbitrário de estados anteriores. A arquitetura desta rede é apresentada na figura 5.7

A cada nova entrada, os valores de ativação são propagados para uma das camadas de cópia, atualizando-a com o atual padrão de entrada. De forma similar, outras camadas são atualizadas com valores anteriores, efetivamente treinando a memória da rede.

Dessa forma, após o treinamento, a rede pode ser utilizada para reconhecer até N padrões de entrada usando as camadas escondidas $p(n)$, sendo $n \leq N$.

5.5.1 Rede LSTM

Uma vez que as redes neurais recorrentes convencionais utilizam o conceito de memória de curto prazo, a eficiência de tais redes para previsões de eventos ocorridos num intervalo de tempo maior frequentemente é baixa. Para solucionar esse problema, uma nova rede foi proposta, conhecida como *Long-Short Term Memory network (LSTM)* [118, 95, 96, 108, 192, 184].

A rede LSTM combina memória de curto prazo, similar à oferecida por redes recorrentes tradicionais, com mecanismos para previsão de longo prazo, visando prever eventos ocorridos dentro de intervalos maiores, mesmo sobre séries temporais com ruído. A rede combina um algoritmo baseado em gradientes com unidades internas específicas, conhecidas como células de memória, para reforçar fluxo constante de erros [118].

Em redes neurais convencionais, o fluxo de erros não é constante, variando ao longo do tempo. Geralmente, o fluxo é calculado por meio da equação 5.35, sendo que o valor do

neurônio de saída k no intervalo de tempo t é representado por $d_k(t)$, o seu sinal de erro por $\theta_k(t)$, e o valor de ativação do neurônio intermediário i é calculado usando a equação 5.36, sendo que sua função de ativação é f_{ij} .

$$\theta_k(t) = f'_k(net_k(t))(d_k(t) - y^k(t)) \quad (5.35)$$

$$y^i(t) = f_i(net_i(t)) \quad (5.36)$$

O valor de entrada do neurônio i é calculado por meio da equação 5.37, considerando que w_{ij} representa o peso da conexão entre os neurônios i e j .

$$net_i(t) = \sum_j w_{ij} y^j(t-1) \quad (5.37)$$

Para os neurônios intermediários j , o sinal de erro retornado (*backpropagated error signal*) é calculado através da equação 5.38.

$$\theta_j(t) = f'_j(net_j(t)) \sum_i w_{ij} \theta_i(t+1) \quad (5.38)$$

A contribuição correspondente ao valor total de peso de w_{ij} é calculada como sendo $\alpha \theta_j(t) y^i(t-1)$, onde α a taxa de aprendizado e l é um neurônio arbitrário conectado a j .

Considerando uma rede com $1..n$ neurônios intermediários, é possível calcular a propagação do erro entre eles. Observando uma conexão entre dois neurônios inter-conectados u e v , o erro observado no instante de tempo t no neurônio u é propagado inversamente (*back into time*) por q intervalos de tempo, para o neurônio v . Com isso, o valor de erro é dimensionado de acordo com a equação 5.39

$$\frac{\delta \theta_v(t-q)}{\delta \theta_u(t)} = \begin{cases} f'_v(net_v(t-1)) w_{uv} & q = 1 \\ f'_v(net_v(t-q)) \sum_{l=1}^n \frac{\delta \theta_l(t-q+1)}{\delta \theta_u(t)} w_{lv} & q > 1 \end{cases} \quad (5.39)$$

Considerando $l_q = v$ e $l_0 = u$, a equação 5.39 pode ser representada como 5.40 [118].

$$\frac{\delta \theta_v(t-q)}{\delta \theta_u(t)} = \sum_{l_1=1}^n \dots \sum_{l_{q-1}=1}^n \prod_{m=1}^q f'_{l_m}(net_{l_m}(t-m)) w_{l_m l_{m-1}} \quad (5.40)$$

A soma de n^{q-1} termos $\prod_{m=1}^q f'_{l_m}(net_{l_m}(t-m)) w_{l_m l_{m-1}}$ determina o fluxo de erro inverso total.

Nos casos em que $|f'_{l_m}(net_{l_m}(t-m)) w_{l_m l_{m-1}}| > 1$ para todos os valores de m , o erro cresce de forma exponencial com a variação de q , resultando em oscilação de pesos e aprendizado instável.

Por outro lado, quando $|f'_{l_m}(net_{l_m}(t-m)) w_{l_m l_{m-1}}| < 1$, o erro decresce de forma exponencial. Com isso, a taxa de aprendizado tende a zero e a rede não tem capacidade de aprendizagem eficiente.

Isso torna o método convencional de aprendizado, baseado em *back-propagation* de erros, sensível a distrações. Para solucionar este problema, *LSTM* mantém o fluxo de erro constante, considerando que o fluxo de erro do neurônio j no intervalo de tempo t é igual a $\theta_j(t) = f'_j(net_j(t)) \theta_j(t+1) w_{jj}$ e satisfazendo a equação 5.41:

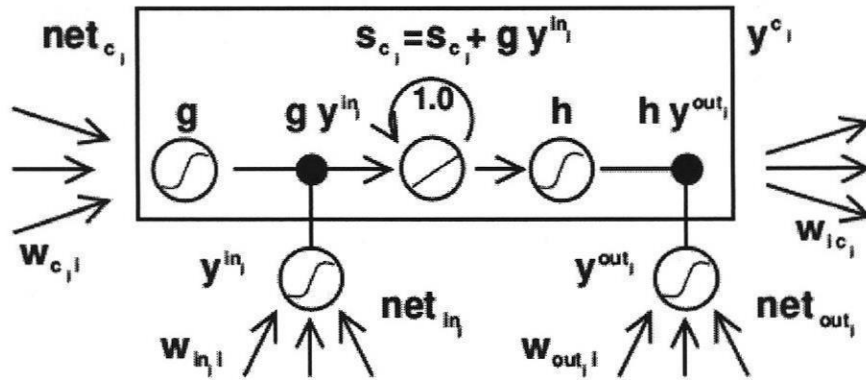


Figura 5.8: Célula de memória de *LSTM*

$$f'_j(net_j(t))w_{jj} = 1 \quad (5.41)$$

Integrando a equação 5.41, é possível ver que $f_j(net_j(t)) = \frac{net_j(t)}{w_{jj}}$. Ou seja, o valor de f_j tem que ser linear, e a ativação do neurônio j constante, conforme a equação 5.42. Essa equação é denominada de circulação constante de erro (*constant error carousel*, ou *CEC*), sendo a principal propriedade da rede *LSTM* [118].

$$y_j(t+1) = f_j(net_j(t+1)) = f_j(w_{jj}y^j(t)) = y^j(t) \quad (5.42)$$

Entretanto, uma vez que os neurônios podem ser modificados ao longo do tempo, é necessário prover mecanismos para prevenir a modificação de pesos por valores de entrada inadequados, e prevenir previsões falsas resultantes de conflitos entre as memórias de curto e longo prazo. Para isso, *LSTM* introduz mecanismos adicionais na rede:

1. Unidade multiplexadora de proteção de entrada (*input gate unit*), utilizada para proteger a memória armazenada no neurônio j das modificações causadas por entradas indesejáveis;
2. Unidade multiplexadora de proteção de saída (*output gate unit*), utilizada para proteger outras unidades da rede das perturbações que podem ser causadas pelo conteúdo armazenado no neurônio j , quando tal conteúdo é irrelevante para elas.

Com isso, o neurônio resultante torna-se mais complexo, e passa a ser denominado de célula de memória, sendo apresentado na figura 5.8.

Conforme pode ser visto na figura 5.8, a j -ésima célula de memória é denominada de c_j . Cada célula de memória é construída em torno de uma unidade central linear auto-conectada e fixa (e.g., *CEC*). Além de net_{c_j} , c_j recebe os valores de entrada da unidade de proteção de entrada in_j , e da unidade de proteção de saída out_j . A ativação de in_j na instante de tempo t é representada por $y^{in_j}(t)$; e a de out_j por $y^{out_j}(t)$, sendo que $y^{out_j}(t) = f_{out_j}(t)$ e $y^{in_j}(t) = f_{in_j}(net_{in_j}(t))$.

Os valores de $net_{out_j}(t)$, $net_{in_j}(t)$ e $new_{c_j}(t)$ são calculados por meio das equações 5.43, 5.44 e 5.45, sendo que o índice somatório u pode representar células de memória, unidades de

proteção de entrada e de saída ou até neurônios convencionais, uma vez que todos eles podem trazer informações relevantes sobre o estado atual da rede.

$$net_{out_j}(t) = \sum_u w_{out_{ju}} y^u(t-1) \quad (5.43)$$

$$net_{in_j}(t) = \sum_u w_{in_{ju}} y^u(t-1) \quad (5.44)$$

$$net_{c_j}(t) = \sum_u w_{c_{ju}} y^u(t-1) \quad (5.45)$$

No intervalo de tempo t , o valor de saída $y^{c_j}(t)$ de c_j é calculado através da equação 5.46, com o estado interno $s_{c_j}(t)$ determinado pela equação 5.47.

$$y^{c_j}(t) = y^{out_j}(t) h(s_{c_j}(t)) \quad (5.46)$$

$$s_{c_j}(t) = \begin{cases} 0 & t = 0 \\ s_{c_j}(t-1) + y^{in_j}(t) g(net_{c_j}(t)) & t > 0 \end{cases} \quad (5.47)$$

A função diferencial g é utilizada para exprimir net_{c-j} , e a função h visa expandir os valores de saída da célula de memória c_j , calculado a partir do estado interno s_{c_j} .

A rede *LSTM* utiliza as unidades de proteção de entrada e de saída para controlar quando o conteúdo da célula de memória deve ser modificado. A unidade de proteção de entrada in_j é utilizada para decidir se a rede deve modificar ou manter o conteúdo da célula c_j ; e a de proteção de saída out_j é empregada para determinar se outras células podem utilizar o conteúdo de c_j .

Enquanto os sinais de erro, armazenados dentro da célula por meio de *CEC*, não devem ser modificados, os que a célula recebe por meio da sua unidade de proteção de saída podem ser considerados. Para isso, a unidade de proteção de saída é treinada para aprender quais são os erros que devem ser passados para *CEC*, e como eles devem ser escalonados. Da mesma forma, a unidade de proteção de entrada deve ser treinada para aprender quais erros devem ser ignorados. De forma geral, as unidades de proteção são utilizadas para abrir e fechar acesso ao fluxo de erros calculado por *CEC*.

Topologicamente, uma rede *LSTM* geralmente é composta por uma camada de entrada, uma de saída e uma escondida. As células de memória e suas unidades de proteção são localizadas na camada escondida, sendo que ela também pode ter neurônios convencionais conectados tanto às células quanto às unidades de proteção. Todos os neurônios de uma camada, com a exceção das unidades de proteção, são conectados diretamente aos neurônios da camada superior, servindo de entrada para eles.

Além de células de memória, a rede *LSTM* também pode ter blocos de memória. Tais blocos são compostos por células que compartilham a mesma unidade de proteção de entrada e de saída. Dessa forma, S células que compartilham as mesmas unidades de proteção de entrada e de saída formam um bloco de memória de tamanho S . Tais blocos são utilizados para facilitar na troca de informações entre células adjacentes.

Um exemplo da topologia de rede neural *LSTM* é apresentado na figura 5.9 [98].

O processo de aprendizado da rede *LSTM* é similar ao das redes convencionais, com a diferença de que os erros recebidos por *back-propagation* não são propagados para fora das células de memória que os receberam. No lugar disso, tais valores são utilizados pelas unidades

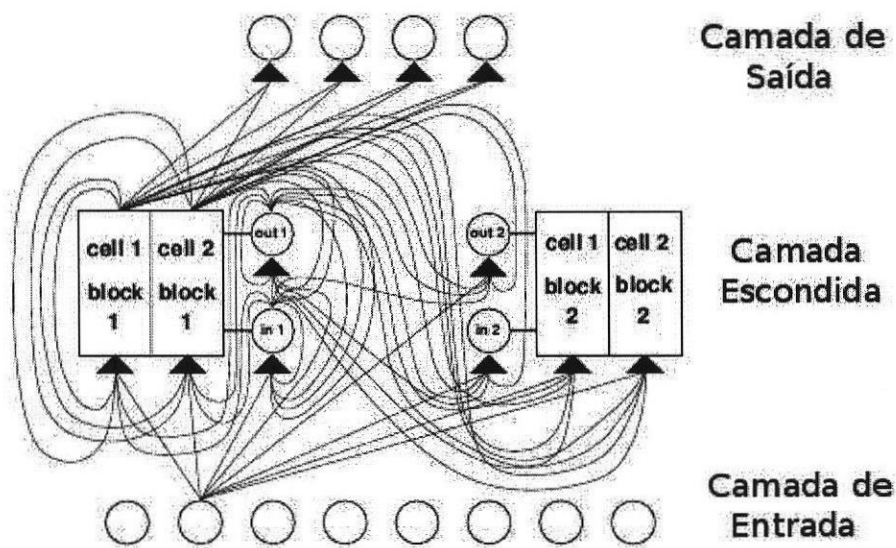


Figura 5.9: Exemplo de *LSTM*

de proteção para determinar se os pesos das conexões da célula devem ser modificados ou mantidos. Da mesma forma, o erro calculado pela célula é utilizado para modificar o seu valor de saída correspondente.

Redes *LSTM* apresentam uma baixa complexidade computacional [118, 96] e taxa de aprendizado significativamente mais rápida de que redes recorrentes convencionais. Além disso, *LSTM* possibilita a realização de previsões eficientes com distâncias superiores às redes convencionais.

Como uma extensão à rede *LSTM* é possível citar o trabalho de Ortiz *et al* [184], que utiliza o filtro de Kalman para melhorar o processo de aprendizado da rede. A rede proposta resulta em uma maior eficiência e custo computacional menor. Além disso, o novo algoritmo resulta em uma previsão mais precisa em ambientes com ruído. Outras extensões são de autoria de Gers *et al*, introduzindo módulos de esquecimento de dados irrelevantes por meio de “forget gates” [98] e inter-conexões entre células independentes (“peephole connections”) [97], resultando em aprendizado mais preciso de diferenças entre seqüências de padrões conhecidos.

5.5.2 Rede neural de Hopfield

A rede neural artificial de Hopfield [120] também pertence à família de redes neurais recorrentes, e foi proposta como memória associativa entre dados de entrada e saída. A associação é realizada com base em uma função de energia de Lyapunov [120], que é aplicada em teoria de sistemas dinâmicos para provar a existência de estados estáveis. Nesses sistemas busca-se provar que a mudança no estado de variáveis de comportamento do sistema dinâmico resulta na queda do valor da função, até que seja atingido um mínimo. O uso dessa função de energia motivou a adoção da rede de Hopfield para a solução de problemas de otimização, nos quais pretende-se obter o mínimo global da função de energia, que representa a melhor solução para

Tabela 5.2: Representação do problema de caixeiro-viajante

0	1	0
0	0	1
1	0	0

certa situação. A superfície da função de energia é armazenada nos pesos dos neurônios da rede.

A função de saída adotada para Hopfield [89] é definida na equação 5.48, onde γ é uma constante denominada parâmetro de ganho e u_i é a entrada da rede (net_i) para o neurônio i . Para $\gamma = 1$ a função é linear de derivada positiva. Conforme o valor de γ aumenta a função comporta-se como uma função passo.

$$v_i = g_i(\gamma u_i) = \frac{1}{2}(1 + \tanh(\gamma u_i)) \quad (5.48)$$

Antes de utilizar a rede de Hopfield, é necessário definir uma função de energia a ser aplicada sobre uma matriz que representa uma solução do sistema. Como um exemplo é possível utilizar o problema do Caixeiro-viajante [89], onde se pretende visitar uma série de cidades com as restrições de: percorrer o menor caminho, não visitar mais que uma vez uma cidade e que todas as cidades devem ser visitadas. Cada solução para o problema é expressa em uma matriz aonde as linhas representam as cidades e as colunas a ordem de visita. Por exemplo, a linha r_0 representa a cidade A , a r_1 a cidade B e assim sucessivamente. Da mesma forma, c_0 representa a primeira cidade a ser visitada, c_1 a segunda e assim sucessivamente.

A figura 5.5.2 apresenta um exemplo de solução para o problema do Caixeiro-viajante. O valor 1 significa a visita a uma cidade, sendo que uma cidade não está sendo visitada mais que uma vez e que todas estão sendo visitadas ao menos uma vez. A primeira visita ocorre à cidade r_2 , depois à r_0 e finalmente à r_1 .

Sobre a matriz que representa uma solução para o problema é aplicada a função de energia, visando desfavorecer estados não desejados. Por exemplo, para desfavorecer mais de uma visita a uma cidade deve-se realizar um somatório de produtos dos valores contidos na matriz para cada cidade de acordo com a equação 5.49, onde X representa uma cidade (linha), i e j índices para percorrer as colunas, v_{Xi} e v_{Xj} os valores contidos na matriz na linha X , colunas i e j , respectivamente. Em um estado estável e atingindo o mínimo global, essa equação deve retornar zero, pois cada cidade é visitada somente uma vez. Se uma cidade for visitada mais vezes, essa equação retorna um valor positivo, o qual deve ser utilizado para decrescer os valores de saída dos neurônios da rede neural. Quando os neurônios oferecem uma saída (matriz tal como apresentada na figura 5.5.2) que satisfaz todas as equações que compõem a função de energia, o sistema finaliza a execução, gerando os resultados da ordem de visita das cidades e distância percorrida.

$$\sum_{X=1}^n \sum_{i=1}^n \sum_{j=1, j \neq i}^n v_{Xi} \cdot v_{Xj} \quad (5.49)$$

A equação 5.49 trata somente de uma das restrições para a solução, sendo que outros termos da função de energia devem ser formulados. A equação 5.50 apresenta o segundo termo que garante que a solução somente será válida se uma cidade for visitada por vez, uma vez que

duas cidades não podem ser visitadas simultaneamente pelo mesmo viajante. Da mesma forma, caso uma coluna da matriz acumule um valor de somatório de produtos igual a zero, a solução proposta satisfaz esse termo da função de energia. Caso o valor seja maior que zero, duas ou mais cidades podem ter sido visitadas simultaneamente pelo mesmo viajante, o que não é permitido e deve ser concluído como solução inválida.

$$\sum_{i=1}^n \sum_{X=1}^n \sum_{Y=1, Y \neq X}^n v_{Xi} \cdot v_{Yi} \quad (5.50)$$

O terceiro termo da função de energia é apresentado na equação 5.51. Esse termo garante que o número de visitas às cidades deve ser igual ao número de cidades (n). Caso seja diferente será computado um valor para diminuir o valor de entrada dos neurônios, gerando, portanto, saídas de menor valor.

$$\left(\sum_{X=1}^n \sum_{i=1}^n v_{Xi} - n \right)^2 \quad (5.51)$$

O quarto e último termo, apresentado na equação 5.52, visa minimizar a distância entre cidades, onde d_{XY} representa a distância entre as cidades X e Y armazenada em outra matriz. Esse termo busca computar e gerar o custo total do caminho passando por todas as cidades e retornando à primeira.

$$\sum_{X=1}^n \sum_{Y=1, Y \neq X}^n \sum_{i=1}^n d_{XY} v_{Xi} (v_{Y,i+1} + v_{Y,i-1}) \quad (5.52)$$

Os termos apresentados nas equações 5.49, 5.50, 5.51 e 5.52 compõem a função de energia 5.53 que deve ser minimizada, onde A , B , C e D representam a relevância do termo para a equação.

$$\begin{aligned} E = & A \cdot \sum_{X=1}^n \sum_{i=1}^n \sum_{j=1, j \neq i}^n v_{Xi} \cdot v_{Xj} \\ & + B \cdot \sum_{i=1}^n \sum_{X=1}^n \sum_{Y=1, Y \neq X}^n v_{Xi} \cdot v_{Yi} \\ & + C \cdot \left(\sum_{X=1}^n \sum_{i=1}^n v_{Xi} - n \right)^2 \\ & + D \cdot \sum_{X=1}^n \sum_{Y=1, Y \neq X}^n \sum_{i=1}^n d_{XY} v_{Xi} (v_{Y,i+1} + v_{Y,i-1}) \end{aligned} \quad (5.53)$$

A rede neural de Hopfield aplica a função de energia sobre uma solução, decresce os valores de entrada dos neurônios, que apresentam uma nova solução a ser avaliada. Esse ciclo é executado até que seja obtida uma solução que não contrarie nenhum dos termos da função de energia.

Por oferecer mecanismos de memória associativa, também pode-se aplicar a rede neural de Hopfield para detectar e prever mudanças no comportamento de aplicações.

5.5.3 Técnica de *Simulated annealing*

Uma das desvantagens da rede neural de Hopfield é o fato de que a solução encontrada pode tender a um mínimo local, que nem sempre representa uma solução aceitável. A técnica de *Simulated Annealing*, por sua vez, funciona de maneira muito similar à rede de Hopfield, porém com o objetivo de encontrar uma solução global para o problema [140]. O nome da técnica foi proposto como uma analogia à técnica de *annealing* em metalurgia, uma técnica que consiste em aquecimento e resfrição controlada de materiais com o objetivo de reduzir os seus defeitos.

A técnica introduz o conceito de *temperatura* do sistema para encontrar mínimos globais. Para isso, a cada iteração do algoritmo, o mecanismo considera os estados vizinhos ao atual s , denominados s' . Durante essa avaliação, é determinada a probabilidade de transição do estado atual para um novo, tendendo para estados com energia menor. A vizinhança é especificada de maneira explícita, podendo apresentar variações para cada problema específico.

A probabilidade de transição de estado é calculada por meio da função $P(e, e', T)$, onde e e e' representam as energias $E(s)$ e $E(s')$ dos estados correspondentes s e s' , e o parâmetro T representa a temperatura global do sistema, alterada com o tempo.

A propriedade essencial desta função é o fato de que a probabilidade de transição P deve ser diferente de zero, com $e' > e$ indicando uma transição para um estado pior (i.e., que apresenta maior energia). Essa característica do sistema é utilizada para evitar a estabilização do sistema num mínimo local.

Por outro lado, quando a temperatura T do sistema é próxima a zero, a função de probabilidade deve tender a zero caso $e' > e$, retornando um valor positivo caso contrário. Isso garante que, para valores pequenos de T , o sistema apresentará a tendência de encontrar as soluções com menor energia com o passar de tempo.

Dessa maneira, a técnica de *simulated annealing* pode ser utilizada para determinar mudanças do comportamento de forma similar à rede de Hopfield, oferecendo uma solução ótima global para o problema. Para aplicar essa técnica, entretanto, conhecimentos sobre o funcionamento do sistema são necessários para determinar as variações de temperatura em busca do mínimo global.

5.5.4 Máquinas de Boltzmann

Sob o nome de máquinas de Boltzmann entende-se uma família de redes neurais estocásticas recorrentes, baseadas na técnica de *simulated annealing* [90, 207, 206, 106].

As máquinas de Boltzmann são similares à rede neural de Hopfield, compostas por uma rede de neurônios utilizados para determinar a função de energia do sistema. A principal diferença com a rede de Hopfield é o fato das máquinas de Boltzmann utilizarem neurônios estocásticos no lugar de discretos, usados na técnica de Hopfield.

A função de energia global E é determinada por meio da equação 5.54, onde w_{ij} é o peso da conexão entre os neurônios i e j , $s_j \in 0, 1$ representa o estado do neurônio j e θ_i é o valor de corte (*threshold*) para o neurônio i .

$$E = - \sum_{i < j} w_{ij} s_i s_j + \sum_i \theta_i s_i \quad (5.54)$$

As conexões utilizadas pelas máquinas de Boltzmann são simétricas, e os neurônios não podem ser auto-conectados, satisfazendo $w_{ii} = 0, \forall i$ e $w_{ij} = w_{ji}, \forall i, j$. Dessa forma, é pos-

sível determinar a diferença na energia global do sistema ΔE_i ocorrida quando um neurônio i passa de 0 a 1 por meio da equação 5.55.

$$\Delta E_i = \sum_j w_{ij} s_j - \theta_i \quad (5.55)$$

As máquinas de Boltzmann são compostas por neurônios estocásticos. A probabilidade p_i do neurônio i estar ativo é determinada pela equação 5.56, sendo que T representa a temperatura do sistema, introduzida pela técnica de *simulated annealing*.

$$p_i = \frac{1}{1 + \exp(-\frac{1}{T} \Delta E_i)} \quad (5.56)$$

A arquitetura de máquinas de Boltzmann consiste de neurônios visíveis, que recebem as entradas, e escondidos, utilizados para o cálculo da função de energia. Durante o processo de aprendizagem, os neurônios visíveis processam os valores de entrada e atualizam a temperatura global do sistema. Os escondidos, por sua vez, são atualizados em função dos valores dos neurônios visíveis, visando manter um equilíbrio térmico no sistema.

No processo de aprendizagem, onde V representa o conjunto de entradas no sistema mapeado sobre os neurônios visíveis por meio da função de distribuição $P^+(V)$, a rede calcula uma função de distribuição resultante $P^-(V)$. O objetivo de treinamento é minimizar a distância G entre $P^+(V)$ e $P^-(V)$, calculada por meio da função de Kullback-Leibler, conforme demonstrado na equação 5.57, sendo $v \in V$.

$$G = \sum_v P^+(v) \ln \frac{P^+(v)}{P^-(v)} \quad (5.57)$$

O treinamento consiste de duas fases alternadas. A primeira fase, conhecida como fase positiva, consiste em treinamento dos neurônios visíveis pelos padrões de entrada. Na segunda fase, conhecida como negativa, os valores de entrada não são processados, fazendo com que a rede se adapte aos padrões processados anteriormente. Uma vez que na segunda fase de treinamento a rede funciona de forma automática, como uma máquina, e a energia do sistema é calculada por meio da distribuição de Boltzmann [90, 176], a rede foi nomeada de máquina de Boltzmann.

Entretanto, a rede funciona adequadamente somente para padrões mais simples. Para entradas mais complexas, o tempo necessário para a estabilização da energia da rede e a complexidade das suas conexões aumentam exponencialmente. Além disso, a presença do ruído nas entradas da rede resulta em aprendizado caótico, uma vez que o processo de treinamento não consegue diferenciar o ruído observado das entradas desejadas.

5.6 Redes Associativas Competitivas

A família de redes neurais competitivas é caracterizada pela abordagem de *winner-takes-all* [133], sendo utilizada amplamente para a classificação e predição de padrões.

Como visto na seção 4, é possível classificar o comportamento de aplicações agrupando estados com comportamento semelhante. No modelo *SOM*, o conceito de vizinhança é utilizado para determinar os estados funcionais de aplicações caracterizadas pelo comportamento similar. Uma vez que a classificação é feita de maneira estática, isso não permite prever o

comportamento futuro de tais aplicações. Para solucionar esse desafio, diversas técnicas foram propostas [17, 16, 157, 144, 224], tais como as descritas a seguir.

5.6.1 Modelo SOM temporal

Uma extensão para o modelo SOM foi proposta em [46], denominada Mapas de Kohonen Temporais, ou *TKM* (*Temporal Kohonen Maps*), com o objetivo de separar diversos padrões de entrada de acordo com a sua ordem e contexto de processamento. Dessa forma, além de agrupar e classificar os padrões de entrada de acordo com a vizinhança, pode-se classificar os diversos padrões em função do tempo.

Para isso, o modelo SOM foi modificado com o objetivo de introduzir o conceito de memória de curto prazo, ou *STM* (*Short Term Memory*), utilizada para armazenar o histórico de comportamento para cada elemento do mapa.

O modelo proposto funciona de maneira análoga ao modelo SOM original, sendo que a única diferença consiste na camada de saída. Enquanto o modelo SOM original zera os valores de neurônios da camada de saída após encontrar o neurônio vencedor, o modelo *TKM* ajusta os valores dos neurônios gradativamente. Para isso, é utilizado o processo de deterioração, demonstrado na equação 5.58, onde α representa uma constante de tempo, $V_i(n)$ é o valor de ativação do neurônio i no instante n , $w_i(n)$ referencia o vetor de pesos da unidade i e $x(n)$ é o padrão de entrada.

$$V_i(n) = \alpha V_i(n-1) - (1/2)\|x(n) - w_i(n)\|^2 \quad (5.58)$$

Dessa forma, o modelo permite classificar os padrões de entrada em função de mudanças no seu comportamento. Entretanto, essa abordagem é limitada, uma vez que não existe nenhum mecanismo recorrente de propagação de erros (*backpropagation*), sendo eficiente apenas para padrões de entrada simples.

Para aumentar a eficiência do modelo *TKM*, a técnica denominada de *RSOM* (*Recurrent Self-Organizing Map*) foi proposta [144], introduzindo uma avaliação recorrente da rede. O modelo possibilita ajustar os valores de entrada em função da classificação da rede, calculando as diferenças para cada elemento da rede através da equação 5.59, onde α é coeficiente temporal, análogo ao α do modelo *TKM* descrito acima e $y_i(n)$ representa a diferença vetorial para cada nó do mapa, e os valores de $x(n)$ e $w_i(n)$ são análogos ao modelo *TKM*.

$$y_i(n) = (1 - \alpha)y_i(n-1) + \alpha(x(n) - w_i(n)) \quad (5.59)$$

Para $\alpha = 1$, o modelo *RSOM* se comporta de maneira análoga ao modelo *SOM* convencional. Tendo $\alpha = 0$, todas as unidades do mapa convergem para os valores médios dos padrões de entrada.

5.6.2 Modelo SOM auto-regressivo

Em [157], um método para classificação não-supervisionada é proposto para processamento de sinais, que consiste na utilização de um mapa auto-organizado de neurônios, sendo que cada nó do mapa é representado por vetor auto-regressivo $x(n+1)$, sendo que $x(k)$, $k = 0, 1, \dots$ é uma série temporal discreta de ordem M , representada pela regressão 5.60.

$$x(n+1) = w_0x(n) + w_1x(n-1) + \dots + w_{M-1}x(n-(M-1)) + e(n) \quad (5.60)$$

Dessa forma, é possível representar o vetor de pesos com W , e o histórico temporal do vetor de entrada $x(n+1)$ com $X(n)$, conforme a equação 5.61

$$X(n) = x(n), x(n-1), \dots, x(n-(M-1)) \quad (5.61)$$

Considerando que as colunas das matrizes W e X são compostos por vetores de dimensão M , pode-se prever o próximo elemento a partir de M elementos anteriores através da equação 5.62.

$$x(n+1) = W(n)^T W \quad (5.62)$$

O mapa é auto-organizado de maneira não-supervisionada durante a fase de treinamento de maneira similar ao modelo SOM tradicional, sendo que cada nó do mapa mantém o seu histórico que pode ser utilizado para prever as futuras transições, conforme a equação 5.62.

5.6.3 Rede neural VQTAM

Em [17], foi proposta uma extensão ao modelo SOM denominado *VQTAM (Vector-Quantized Temporal Associative Memory)*, que consiste na utilização de memória associativa aliada à classificação vetorial de dados feita pela rede *SOM*. A utilização de memória associativa permite realizar a classificação de dados em função dos resultados esperados, possibilitando uma predição de ações futuras por meio de mapeamento de vetores de entrada com os valores esperados de saída.

A memória associativa é introduzida ao modelo *SOM* por meio de modificação do vetor de entrada $x(t)$, transformando-o em vetores x e w , conforme a equação 5.63, onde x^{in} representa os dados de entrada, x^{out} os dados de saída esperados, w_i^{in} é o vetor de pesos relacionado aos dados de entrada e w_i^{out} sendo o vetor de pesos correspondente aos de saída esperados.

$$x(t) = \begin{pmatrix} x^{in}(t) \\ x^{out}(t) \end{pmatrix}, \quad w_i(t) = \begin{pmatrix} w_i^{in}(t) \\ w_i^{out}(t) \end{pmatrix} \quad (5.63)$$

Durante a fase de treinamento, o neurônio vencedor é determinado somente em função dos dados de entrada x^{in} , conforme a equação 5.64.

$$i^*(t) = \arg \min i \equiv A\{\|x^{in}(t) - w_i^{in}(t)\|\} \quad (5.64)$$

Para atualizar os pesos, tanto $x^{in}(t)$ quanto $x^{out}(t)$ são empregados, conforme as equações 5.65 e 5.66, onde $\alpha(t)$ representa a taxa de aprendizado.

$$\Delta w_i^{in}(t) = \alpha(t)h(i^*, i; t)[x^{in}(t) - w_i^{in}(t)] \quad (5.65)$$

$$\Delta w_i^{out}(t) = \alpha(t)h(i^*, i; t)[x^{out}(t) - w_i^{out}(t)] \quad (5.66)$$

$$h(i^*, i; t) = \exp\left(-\frac{\|r_i(t) - r_{i^*}(t)\|^2}{2\omega^2(t)}\right) \quad (5.67)$$

Nas equações 5.65 e 5.66, $h(i^*, t; t)$ é a função Gaussiana de vizinhança, apresentada na equação 5.67, sendo que $r_i(t)$ e $r_{i^*}(t)$ representando as localizações dos neurônios i e i^* no vetor de saída. Os parâmetros $\alpha(t)$ e $\omega(t)$ decrescem de exponencialmente em função do tempo, conforme as equações 5.68 e 5.69, sendo que α_0 e ω_0 representam os seus valores iniciais e α_T e ω_T os valores finais, após T iterações.

$$\alpha(t) = \alpha_0 \left(\frac{\alpha_T}{\alpha_0} \right)^{t/T} \quad (5.68)$$

$$\omega(t) = \omega_0 \left(\frac{\omega_T}{\omega_0} \right)^{t/T} \quad (5.69)$$

Com a rede treinada, pode-se prever o valor de saída estimado de maneira eficiente a partir do sub-vetor w_i^{out} do vetor de entrada, de acordo com a equação 5.70.

$$y(t+1) = w_i^{out}(t) \quad (5.70)$$

5.7 Predição por meio de atrasos no tempo: TDNN

Um grupo de redes neurais inicialmente introduzido por Waibel [229] e conhecido como *TDNN* (*Time-Delay Neural Networks*) é caracterizado pela utilização de uma topologia especial, voltada à reconhecimento de sub-padrões independentes dentro de um padrão de entrada contínuo.

Originalmente, a rede *TDNN* foram propostas para reconhecimento de som. Para o projeto da rede, as seguintes considerações foram feitas:

1. A rede é composta por múltiplas camadas e conexões, e é focada em aprendizado não-linear;
2. A rede deve ser capaz de representar relacionamentos entre eventos em tempo;
3. O número de pesos na rede deve ser suficientemente pequeno comparando com a quantidade de dados treinados. Dessa forma, a rede é forçada a codificar os dados observados, extraindo regularidades e comportamentos repetitivos.

A técnica *TDNN* consiste na introdução de *atrasos* nos padrões de entrada da rede, sendo geralmente composta por uma camada de entrada, uma ou mais camadas escondidas e uma camada de saída, conforme a figura 5.10. Um procedimento de aprendizado recorrente (*back-propagation*) é adotado, utilizando o cálculo de soma média de erros para a determinação do erro.

Em redes neurais convencionais, um neurônio é responsável por calcular uma soma ponderada de todas as suas entradas e, após isso, a propagá-la por meio de uma função não-linear (geralmente função de corte ou sigmóide) para as próximas unidades. Na rede *TDNN*, as entradas são modificadas para introduzir n atrasos à entrada de dados. Dessa maneira, uma camada de entrada composta por y elementos gera $z = y \cdot (n+1)$ entradas para a rede. Dessa forma, a rede necessita de z pesos para y entradas.

Enquanto em redes neurais convencionais cada neurônio geralmente conecta-se somente uma vez com outro da camada sucessora, no modelo *TDNN* são realizadas diversas conexões com pesos independentes. Antes do cálculo do peso de cada conexão, os dados sofrem um

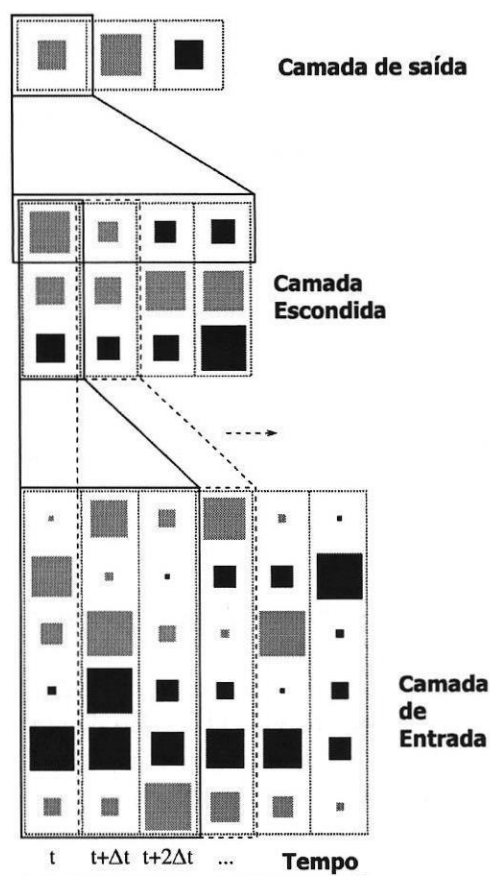


Figura 5.10: Rede *TDNN*

atraso que os elimina do cálculo da soma de pesos. Uma vez que cada conexão é associada com um atraso correspondente, as conexões atuam como o mecanismo de memória de curto prazo, possibilitando relatar e comparar o vetor de entrada atual aos dados observados anteriormente.

As saídas de cada camada são computadas e utilizadas como valores de entrada para as camadas sucessoras, até chegar na camada de saída. Os valores de saída são comparados aos valores esperados, calculando o erro, que é propagado pela rede, atualizando os pesos e, conseqüentemente, decrementando o erro global. O procedimento é repetido até a rede convergir para a saída desejada. Dessa maneira, uma predição eficiente somente é possível após o treinamento repetitivo da rede.

Entre as extensões ao modelo *TDNN* pode-se citar a rede *ATNN* (*Adaptive Time-Delay Neural Network*), que possibilita adaptar os *lags* da rede dinamicamente [62].

5.8 Aprendizado supervisionado: *Reservoir computing*

A família de técnicas conhecidas como *Reservoir Computing* foi introduzida recentemente [222, 126], sendo representada por técnicas como *Echo State Networks (ESN)* [222, 126], *Liquid-state machines* [126, 223] e técnica de *Backpropagation-decorrelation* [126].

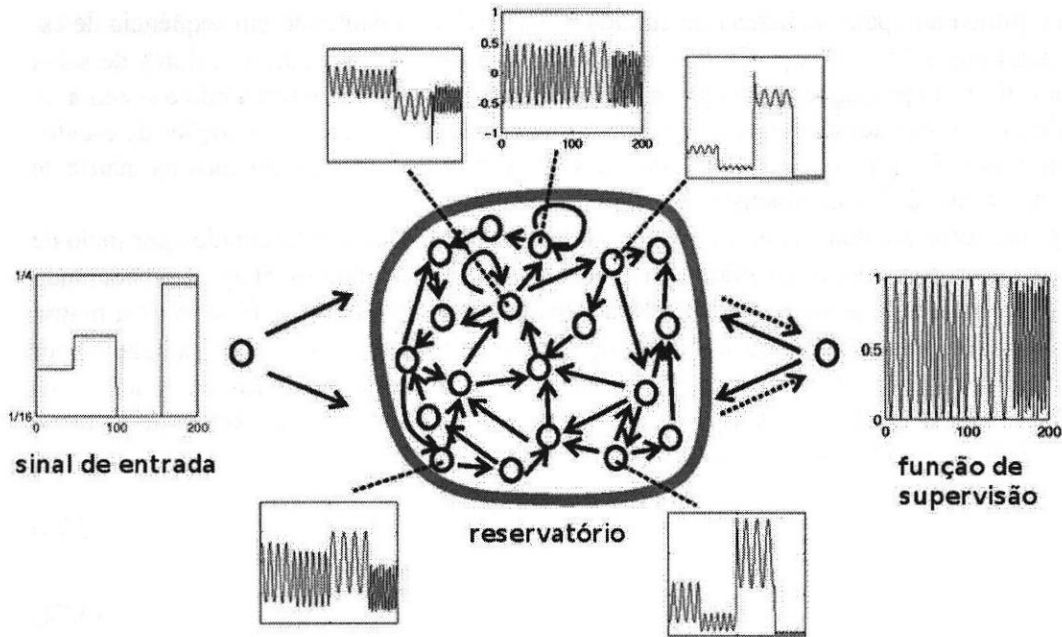


Figura 5.11: Arquitetura de *Reservoir Computing*

A ideia principal compartilhada entre essas abordagens é a existência de um sistema dinâmico recorrente, conhecido como reservatório (*reservoir*), e composto por um número alto de nós interconectados com conteúdo aleatório. Uma função de supervisão é utilizada para treinar o sistema, mapeando os resultados desejados de acordo com a dinâmica interna do reservatório, conforme pode ser visto na figura 5.11.

Conforme pode ser visto na figure 5.11, o mecanismo de treinamento visa mapear N padrões de entrada $u(t)$ de acordo com a saída esperada $y(t)$ em uma sequência $D = [(u(1), y(1)), \dots, (u(N), y(N))]$. Para representar a correlação do padrão de entrada $u(n)$ com o estado interno do reservatório $x(n)$ são utilizados estados estendidos $z(n) = [x(n); u(n)]$.

Durante a execução do reservatório com N unidades, K entradas e L saídas, a atualização dos estados internos é controlada pela equação 5.71, sendo $x(n)$ o estado N -dimensional do reservatório, f uma função sigmóide, W é a matriz de pesos do reservatório de dimensão $N \times N$, W_{in} a matriz de entradas $K \times N$ -dimensional, $u(n)$ é o padrão de entrada K -dimensional, W_{fb} é a matriz de pesos da função de saída com dimensão $L \times N$ e $y(n)$ é o sinal de saída L -dimensional no intervalo de tempo n .

$$y(n+1) = f(Wx(n) + W_{in}u(n+1) + W_{fb}y(n)) \quad (5.71)$$

Após o treinamento, a execução do sistema é feita por meio da equação 5.72, sendo W_{out} é matriz dos pesos de saída de dimensão $(K + N) \times L$, g a função de ativação e $z(n)$ o estado estendido do sistema no intervalo n .

$$y(n) = g(W_{out} * z(n)) \quad (5.72)$$

Como pode ser visto a partir das equações 5.71 e 5.72, durante o treinamento inicial o

sistema é fomentado pela sequência de entrada $u(1), \dots, u(N)$, resultando em sequência de estados estendidos $z(1), \dots, z(N)$. Caso a função de supervisão é utilizada, os valores de saída esperados $d(n)$ são propagados para as unidades de saída, efetivamente treinando o sistema. A matriz de estados estendidos resultante é transformada em uma matriz de transições de estados S de dimensão $N * (N + K)$, e os valores de saída desejados são armazenados na matriz de supervisão D , que é $N * L$ -dimensional.

Os pesos correspondentes aos valores de saída desejados W_{out} são calculados por meio de regressão linear dos valores de saída $d(n)$ sobre os estados estendidos $z(n)$. Considerando $R = S'S$ como sendo a matriz de correlação dos estados estendidos, e $P = S'D$ a matriz de co-correlação dos estados com as saídas desejadas, os valores de W_{out} são calculados por meio de equações 5.73 ou 5.74, onde S^t é a matriz pseudo-inversa de S . Enquanto ambos os métodos podem ser aplicados, a equação 5.73 requer tempo de execução significativamente menor, e 5.74 apresenta resultados estáveis.

$$W_{out} = R^{-1}P \quad (5.73)$$

$$W_{out} = (S^t D)' \quad (5.74)$$

A maior desvantagem do modelo de *Reservoir Computing* é a complexidade espacial da abordagem, que cresce exponencialmente em função do número de padrões avaliados [126].

5.9 Aprendizado controlado: SRNN

SRNN é uma rede neural recorrente proposta recentemente [94] COM O objetivo de predição de padrões *on-line* eficiente, avaliando o comportamento atual do sistema. A estrutura de rede é similar à utilizada pelas técnicas de *Reservoir Computing*, com a diferença de que uma rede neural recorrente completamente interconectada é utilizada no lugar do reservatório. A técnica evita problemas de estabilidade, observados nas técnicas de *Reservoir Computing* parametrizando os pesos da camada escondida da rede de forma a limitar o espectro dos *eigenvalues* dos seus pesos.

A camada escondida da rede *SRNN* é composta por neurônios interconectados circularmente, conforme pode ser visto na figura 5.12, sendo que todos os neurônios-vizinhos compartilham o mesmo peso β_1 ; os vizinhos-dos-vizinhos compartilham o peso β_2 e assim por diante. Dessa maneira, a matriz de pesos resultante é organizada conforme a equação 5.75.

$$\begin{pmatrix} 0 & \beta_{N-1} & \dots & \beta_N \\ \beta_1 & 0 & \dots & \dots \\ \dots & \dots & 0 & \beta_{N-1} \\ \beta_{N-1} & \dots & \beta_1 & 0 \end{pmatrix} \quad (5.75)$$

Para padrões de entrada d -dimensionais, a camada escondida é replicada d vezes, sendo que todos os neurônios de entrada e de saída são conectados a todos os elementos escondidos.

O funcionamento da rede é similar ao *Reservoir Computing* sendo controlado pelas equações 5.76 e 5.77, sendo b_1 e b_2 vetores de controle, W_{hid} , W_{in} e W_{out} as matrizes de interconexões entre as camadas, g e h as funções de ativação e s_t e x_t os vetores de camada escondida e de entrada no intervalo de tempo t .

- $Pr(B)$ é a probabilidade anterior do evento B , que atua como uma constante de normalização;
- α é a constante de normalização.

Dessa maneira, o teorema pode ser re-escrito de maneira informal, conforme a equação 5.79, onde P representa a probabilidade posterior, S a semelhança entre a probabilidade atual e a anterior, A a probabilidade anterior e C a constante de normalização.

$$P = \frac{S * A}{C} \quad (5.79)$$

Em outras palavras, é possível definir a probabilidade posterior como sendo proporcional à sua semelhança à probabilidade anterior.

Além disso, a relação $Pr(B|A)/Pr(B)$ pode ser denominada de semelhança padronizada. Dessa forma, o teorema pode ser re-formulado conforme a equação 5.80, onde P representa a probabilidade posterior, S a semelhança normalizada entre probabilidade anterior e a atual e A a probabilidade anterior.

$$P = S * A \quad (5.80)$$

A probabilidade do acontecimento de eventos é utilizado no modelo de Redes Bayesianas (*Bayesian network*, ou *Bayesian Belief Network*), sendo um modelo probabilístico gráfico que representa as independências probabilísticas entre um conjunto de variáveis [29, 48, 238, 114, 160, 105]. As variáveis podem ser independentes ou compostas por seqüências contínuas, como, por exemplo, sinais de voz ou seqüências de proteínas [165]. Neste caso, a rede é conhecida como rede Bayesiana dinâmica [136, 183].

Formalmente, redes Bayesianas são representados por grafos acíclicos, cujos nós representam as variáveis e arestas codificam a sua independência condicional, conforme mostrado na figura 5.13, que apresenta uma rede Bayesiana composta pelos estados A e B , sendo que a relação entre eles é determinada pela sua conexão. Caso o arco que os interconecta parte do estado A , tal estado é definido como estado-pai, sendo B o estado-filho de A . Dessa forma, um conjunto de estados que levam a um estado X_i é descrito como sendo $parents(X_i)$. Um grafo acíclico direcionado é considerado uma rede Bayesiana relativa a um conjunto de variáveis, quando a distribuição conjunta dos seus estados pode ser descrita por meio do produto da distribuições locais dos estados e seus respectivos estados-pais, conforme mostrado na equação 5.81.

$$P(X_1, \dots, X_n) = \prod_{i=1}^n P(X_i | parents(X_i)) \quad (5.81)$$

Quando um estado X_i não tem nenhum estado-pai, a sua distribuição de probabilidade é considerada incondicional. Caso contrário, ela é considerada condicional.

Para representar relações indiretas entre os estados de uma rede Bayesiana, geralmente é utilizado o modelo de árvore de Chow-Liu [141, 122], que calcula a probabilidade aproximada de distribuição conjunta $P(X_1, X_2, \dots, X_n)$ como um produto das distribuições condicionais e marginais de segunda ordem. Por exemplo, uma distribuição da 6a dimensão $P(X_1, X_2, X_3, X_4, X_5, X_6)$, representada por meio a árvore apresentada na figura 5.14, pode

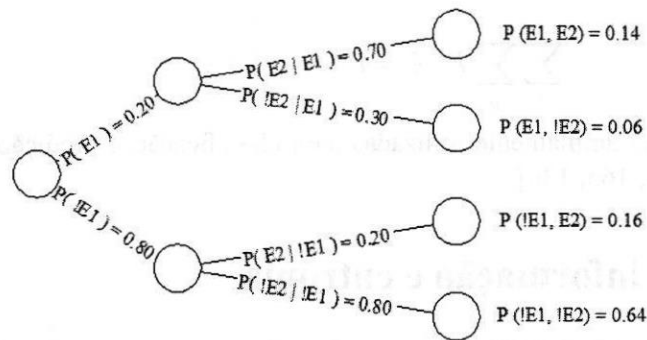


Figura 5.13: Exemplo de rede Bayesiana

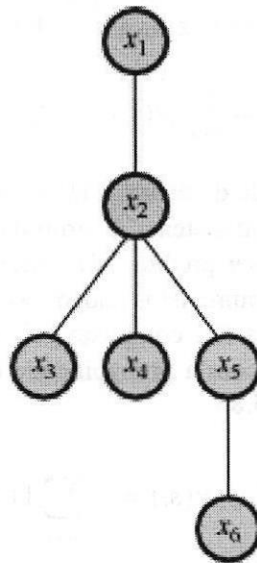


Figura 5.14: Exemplo de árvore de dependências da 2a ordem

ser aproximada conforme a equação 5.82, onde cada termo do produto introduz uma nova variável.

$$P'(X_1, X_2, X_3, X_4, X_5, X_6) = P(X_6|X_5)P(X_5|X_2)P(X_4|X_2)P(X_3|X_2)P(X_2|X_1)P(X_1) \quad (5.82)$$

De forma geral, a probabilidade de distribuição conjunta de variáveis independentes e discretas X e Y pode ser calculado através da equação 5.83, sendo que as suas variáveis satisfazem a equação 5.84.

$$P(X = x, Y = y) = P(Y = y|X = x)P(X = x) = P(X = y|Y = y)P(Y = y) \quad (5.83)$$

$$\sum_x \sum_y P(X = x, Y = y) = 1 \quad (5.84)$$

Redes Bayesianas são amplamente utilizadas para classificação e predição do comportamento de processos [183, 165, 136].

5.11 Teoria de informação e entropia

A teoria da informação é um ramo da teoria da probabilidade e da matemática estatística que visa estimar a capacidade de um sistema gerar nova informação (ou *bits* de informação) [89]. Para quantificar os *bits* de informação produzidos por um sistema, foi introduzido o conceito de *entropia* (equação 5.85, onde $H(S)$ representa a entropia do sistema S , composto por conjunto de eventos $s_i, i = 1..q$, sendo que $P(s_i)$ representa a probabilidade do evento i acontecer).

$$H(S) = - \sum_{i=1}^q P(s_i) \log_2 P(s_i) \quad (5.85)$$

A entropia representa a probabilidade de ocorrência de eventos inesperados. Um sistema com entropia igual a zero representa um sistema determinístico, onde as probabilidades de ocorrência de todos os eventos podem ser preditas. Da mesma forma, um sistema com alta entropia significa que diversos eventos imprevistos podem ocorrer.

Por exemplo, supondo que o sistema S é composto pelo evento s_1 , com a probabilidade $P(s_1) = 1$ (isto é, o evento s_1 ocorre com a probabilidade de 100%), é possível calcular a entropia do sistema através da equação 5.86.

$$H(S) = - \sum_{i=1}^1 P(s_i) \log_2 P(s_i) = - \sum_{i=1}^1 1 * \log_2(1) = 0 \quad (5.86)$$

Nesse caso, a entropia do sistema é igual a zero, ou seja, o sistema é completamente determinístico.

Da mesma forma, se o sistema S for composto por estados s_1 e s_2 , com probabilidades $P(s_1) = 0.1$ e $P(s_2) = 0.2$, a entropia do sistema é calculada pela equação 5.87.

$$\begin{aligned} H(S) &= - \sum_{i=1}^2 P(s_i) \log_2 P(s_i) = \\ &= -((P(s_1) * \log_2 P(s_1)) + (P(s_2) * \log_2 P(s_2))) = \\ &= -((0.9 * \log_2(0.9)) + (0.1 * \log_2(0.1))) = 0.469 \end{aligned} \quad (5.87)$$

Dessa forma, a entropia do sistema S passa a ser igual a 0.469.

A entropia pode ser utilizada para determinar variações no comportamento de processos. Considerando, por exemplo, o processo apresentado pela cadeia de Markov na figura 5.15.a, é possível definir a entropia do sistema (equação 5.89) como a soma das probabilidades de transições entre estados, de acordo com a equação 5.85, onde $P(s_i)$ é a probabilidade de transição de estados.

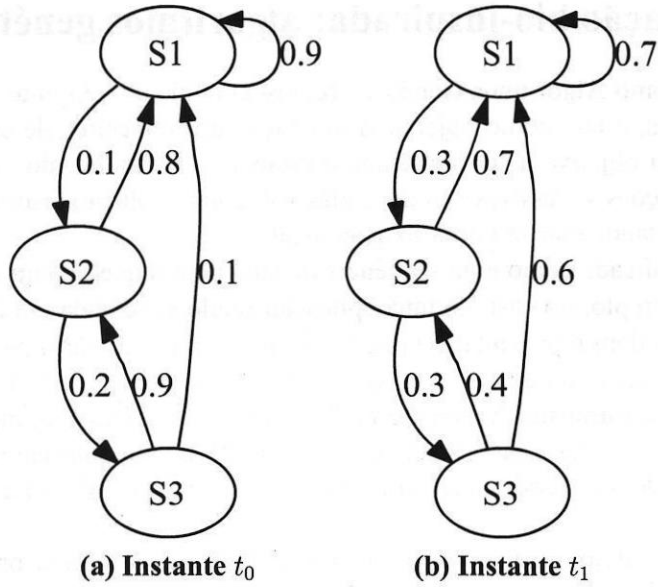


Figura 5.15: Cadeia de Markov para cálculo de entropia

$$\begin{aligned}
 H(S) &= - \sum_{i=1}^6 P(s_i) \log_2 P(s_i) = \\
 &= -((0.9 * \log_2(0.9)) + (0.1 * \log_2(0.1)) + (0.2 * \log_2(0.2)) + \\
 &\quad (0.8 * \log_2(0.8)) + (0.1 * \log_2(0.1)) + (0.9 * \log_2(0.9))) = \\
 &= -(-0.14 - 0.33 - 0.46 - 0.36 - 0.33 - 0.14) = 1.76
 \end{aligned} \tag{5.88}$$

Nesse caso, a entropia do sistema é igual a 1.76. Considere, agora, que no decorrer da execução do sistema, a transição do estado S_1 para S_2 foi alterada conforme figura 5.15.b. Nesse caso a entropia do sistema passa a ser representada pela equação 5.90, que é igual a 2.73.

$$\begin{aligned}
 H(S) &= - \sum_{i=1}^6 P(s_i) \log_2 P(s_i) = \\
 &= -((0.7 * \log_2(0.7)) + (0.3 * \log_2(0.3)) + (0.3 * \log_2(0.3)) + \\
 &\quad (0.7 * \log_2(0.7)) + (0.6 * \log_2(0.6)) + (0.4 * \log_2(0.4))) = \\
 &= -(-0.36 - 0.52 - 0.52 - 0.36 - 0.44 - 0.53) = 2.73
 \end{aligned} \tag{5.89}$$

A alteração de entropia do sistema do instante t_0 para t_1 é igual a 0.97, o que confirma que o sistema tornou-se mais propício para a ocorrência de eventos inesperados. Pode-se concluir que quanto maior é a probabilidade de prever futuras transições entre estados, menor é a entropia do sistema.

5.12 Computação bio-inspirada: algoritmos genéticos

A técnica conhecida como Algoritmos Genéticos representa um sub-conjunto da classe de algoritmos combinatórios, e tem como objetivo a utilização dos conceitos de evolução natural [11]. Dessa forma, um objetivo a ser alcançado é estabelecido (conhecido como *função fitness*), e diversas operações sucessivas são aplicadas sobre uma solução arbitrária (conhecida como *cromossomo*) tentando chegar numa solução ideal.

Uma solução é codificada como uma sequência de parâmetros que podem ser alterados ou recombinados. Por exemplo, no caso de uma aplicação sendo executada em ambiente distribuído, os parâmetros podem representar o tamanho de memória necessário para o seu funcionamento, uso de recursos computacionais, largura de banda necessária, etc. Um determinado conjunto desses atributos forma um “cromossomo”, ou “indivíduo”. Um conjunto de cromossomos é conhecido como uma “geração”, sendo que cada alteração nos seus componentes forma novas gerações-descendentes. Cada geração é formada por uma população com um número finito de indivíduos.

Para cada geração é determinada a aproximação dela à função *fitness* procurada, sendo que os melhores indivíduos da população são utilizados para gerar as gerações sucessivas e os piores são descartados.

Para gerar uma próxima geração, pode-se aplicar operações como:

Tabela 5.3: Operação de cruzamento	
$X_1 X_2 X_3 X_4 X_5 X_6$	$X_1 X_2 Y_3 Y_4 Y_5 Y_6$
$Y_1 Y_2 Y_3 Y_4 Y_5 Y_6$	$Y_1 Y_2 X_3 X_4 X_5 X_6$

(a) Antes do cruzamento (b) Depois do cruzamento

- Cruzamento (*cross-over*): nessa operação, os atributos dos cromossomos são permutados, formando novos indivíduos. Um exemplo dessa operação é mostrado na figura 5.12.
- Mutação (*mutation*): essa operação altera de maneira aleatória alguns dos atributos dos indivíduos com o objetivo de procurar novos caminhos até a solução. Ela pode ser extremamente relevante caso todos os indivíduos apresentem parâmetros semelhantes, ou a solução atual se estabeleceu num mínimo local.
- Predação, *niching*, *crowding*: essas operações são utilizadas para remover indivíduos da população e criar sub-populações, visando localizar novos caminhos até o resultado esperado.
- entre outras.

Como as vantagens dos algoritmos genéticos é possível citar a facilidade de implementação e uso, sua eficácia, sendo possível alcançar uma solução ideal durante a execução do algoritmo, e aplicabilidade na maioria de tarefas computacionais. Como desvantagens, entretanto, é necessário nomear alto custo operacional decorrente da necessidade de re-combinação contínua de diversos indivíduos em larga escala.

Dessa forma, mesmo com a possibilidade de prover resultados ótimos, os algoritmos genéticos freqüentemente não são utilizados devido a um alto tempo necessário para chegar no resultado.

5.13 Considerações finais

Neste capítulo foram apresentadas diversas técnicas de predição do comportamento de processos, descrevendo abordagens tanto estatísticas quanto as baseadas em técnicas de inteligência artificial.

Entre as técnicas estatísticas, técnicas baseadas no modelo de cadeias de Markov, abordagens de auto-regressão linear e não-linear e técnicas que empregam a teoria de Bayes foram apresentadas. Estas técnicas, embora sejam eficientes para alguns casos, não são adequados para ambientes com ruído. Uma solução para ambientes com ruídos, filtro de Kalman, também foi apresentada.

Como uma das formas de medir a eficiência da predição e representação do comportamento de processo foi apresentada, discutindo a teoria de informação e entropia.

Além disso, redes neurais preditoras, tais como redes recorrentes, rede de Hopfield, máquinas de Boltzmann e rede *TDNN*, voltadas para predições a curto prazo. Um modelo para predição tanto de curto quanto de longo alcance também foi apresentado, sendo introduzido pela rede *LSTM*.

Técnicas mais modernas, como *Reservoir Computing* e *Spiral Neural Network*, também foram abordadas, junto com métodos baseados na computação evolutiva.

Capítulo 6

Conclusões

Neste documento foram abordadas diversas técnicas de extração, classificação e predição do comportamento de processos, enfatizando ambientes e sistemas distribuídos.

O objetivo deste documento não foi providenciar uma descrição completa e detalhada das técnicas, mas abordar as suas principais características, vantagens, desvantagens e possíveis casos de uso. Enquanto o nosso foco foi voltado à processos distribuídos, todas os métodos abordados podem ser utilizados em mais diversas áreas computacionais.

A área de descrição e predição do comportamento de processos encontra-se em constante evolução no decorrer das últimas décadas, como é possível ver pelos trabalhos estudados, e novas abordagens são introduzidas constantemente. Com isso, enquanto é possível notar que tais técnicas se tornam cada vez mais avançadas e eficientes, não tem nenhuma solução universal que se aplicaria a todos os problemas.

A escolha da técnica mais adequada, portanto, tem que levar em consideração um estudo sobre o comportamento esperado, uma vez que abordagens distintas pode ser mais eficiente em casos específicos.

Em função disso, o nosso objetivo é a atualização constante deste documento, incluindo novas técnicas, abordagens e métodos na medida do possível.

Referências Bibliográficas

- [1] D. ABRAMSON, R. BUYYA, and J. GIDDY. A computational economy for grid computing and its implementation in the nimrod-g resource broker. citeseer.ist.psu.edu/abramson02computational.html, 2002.
- [2] J. Adibi and W. M. Shen. Self-similar layered hidden markov models. In *PKDD '01: Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 1–15, London, UK, 2001. Springer-Verlag.
- [3] I. Ahrns, J. Bruske, and G. Sommer. On-line learning with dynamic cell structure. In F. Fogelman-Soulié and P. Gallinari, editors, *Proc. ICANN'95, International Conference on Artificial Neural Networks*, volume II, pages 141–146, Nanterre, France, 1995. EC2.
- [4] M.B. Al-Daoud, N.B. Venkateswarlu, and S.A. Roberts. Fast k-means clustering algorithms. citeseer.ist.psu.edu/al-daoud95fast.html.
- [5] D. Alahakoon, S. K. Halgamuge, and B. Srinivasan. Self growing cluster development approach to data mining. *IEEE International Conference on Systems, Man, and Cybernetics*, 3:2901–2906, 1998.
- [6] M. K. Albertini and R. F. Mello. A self-organizing neural network for detecting novelties. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 462–466, New York, NY, USA, 2007. ACM Press.
- [7] C. Amerijckx, M. Verleysen, P. Thissen, and J. D. Legat. Image compression by self-organized Kohonen map. *IEEE Transactions on Neural Networks*, 9(3):503–507, 1998.
- [8] C. Amza, A. L. Cox, S. Dwarkadas, and W. Zwaenepoel. Software DSM protocols that adapt between single writer and multiple writer. In *Proc. of the 3rd IEEE Symp. on High-Performance Computer Architecture (HPCA-3)*, pages 261–271, 1997.
- [9] A. C. Arpaci-Dusseau, D. E. Culler, and M. Mainwaring. Scheduling with Implicit Information in Distributed Systems. In *Proceedings of ACM SIGMETRICS'98*, pages 233–248, August 1998.
- [10] M. Arunachalam, A. Choudhary, and B. Rullman. Implementation and evaluation of prefetching in the Intel Paragon Parallel File System. In *Proceedings of the Tenth International Parallel Processing Symposium*, pages 554–559, April 1996.
- [11] T. Back. Optimization by means of genetic algorithms. In E. Köhler, editor, *36th International Scientific Colloquium*, pages 163–169, Technical University of Ilmenau, 1991.

- [12] M. Baker. Cluster computing white paper. <http://www.dcs.port.ac.uk/~mab/tfcc/WhitePaper/WhitePaper.htm>, 2000.
- [13] S. Baluja and S. Fahlman. Reducing network depth in the cascade-correlation learning architecture. Technical Report CMU-CS-94-209, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, October 1994.
- [14] A. Barak and A. Braverman. Memory ushering in a scalable computing cluster. In *Proc. IEEE Third Int. Conf. on Algorithms and Architecture for Parallel Processing*, 1997.
- [15] A. Barak and O. La'adan. The MOSIX multicomputer operating system for high performance cluster computing. *Future Generation Computer Systems*, 13(4-5):361-372, 1998.
- [16] G. A. Barreto and A. F. R. Araújo. A self-organizing narx network and its application to prediction of chaotic time series. In *Proc. of the IEEE-INNS Int. Joint Conf. on Neural Networks, (IJCNN'01)*, volume 3, pages 2144-2149, Washington, D.C., 2001.
- [17] G. A. Barreto and A. F. R. Araújo. Identification and control of dynamical systems using the self-organizing map. In *Special issue of IEEE TNN on temporal coding*, pages 1244-1259, 2004.
- [18] B. Barrett, J. Squyres, and A. Lumsdaine. Integration of the LAM/MPI environment and the PBS scheduling system. In *Proceedings, 17th Annual International Symposium on High Performance Computing Systems and Applications*, Quebec, Canada, May 2003.
- [19] R. Barve, M. Kallahalla, P. J. Varman, and J. S. Vitter. Competitive parallel disk prefetching and buffer management. In *Proceedings of the Fifth Workshop on Input/Output in Parallel and Distributed Systems*, pages 47-56, San Jose, CA, November 1997. ACM Press.
- [20] T. Bayes. An essay towards solving a problem in the doctrine of chances. In *Philosophical Transactions of the Royal Society of London*, www.stat.ucla.edu/history/essay.pdf, 1763.
- [21] A. J. Bell and T. J. Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural Computation*, 7(6):1129-1159, 1995.
- [22] B. N. Bershad and M. J. Zekauskas. Midway: Shared memory parallel programming with entry consistency for distributed memory multiprocessors. Technical Report CMU-CS-91-170, Carnegie Mellon University, Pittsburgh, PA (USA), 1991.
- [23] G. Bianchi and I. Tinnirello. Kalman filter estimation of the number of competing terminals in an ieee 802.11 network. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, volume 2, pages 844-852 vol.2, 2003.
- [24] R. Bianchini, L. I. Kontothanassis, R. Pinto, M. De Maria, M. Abud, and C. L. Amorim. Hiding communication latency and coherence overhead in software DSMs. In *Proc. of the 7th Symp. on Architectural Support for Programming Languages and Operating Systems (ASPLOS VII)*, pages 198-209, 1996.

- [25] R. Bianchini, R. Pinto, and C. L. Amorim. Data prefetching for software dsms. In *Proceedings of the 12th international conference on Supercomputing*, pages 385–392. ACM Press, 1998.
- [26] S. A. Billings and G. L. Zheng. Radial basis function network configuration using genetic algorithms. *Neural Networks*, 8(6):877–890, 1995.
- [27] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. Wiley-Interscience, New York, NY, USA, 1998.
- [28] A. G. Bors and I. Pitas. Median radial basis function neural network. *IEEE Transactions on Neural Networks*, 7(6):1351–64, 1996.
- [29] R. Bouckaert. Properties of measures for bayesian belief network learning. In *10th Conf. Uncertainty in Artificial Intelligence, Seattle, WA, 1994.*, 1994.
- [30] G. E. P. Box and G. Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.
- [31] P. S. Bradley, U. M. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Knowledge Discovery and Data Mining*, pages 9–15, 1998.
- [32] T. Brecht and K. Guha. Using Parallel Program Characteristics in Dynamic Processor Allocation Policies. *Performance Evaluation*, 27/28(4):519–539, 1996.
- [33] G. Burns, R. Daoud, and J. Vaigl. LAM: An Open Cluster Environment for MPI. In *Proceedings of Supercomputing Symposium*, pages 379–386, 1994.
- [34] V. Burzevski and C. Mohan. Hierarchical growing cell structures. In *ICNN96: Proc. Int'l. Conf. Neural Networks*, June 1996.
- [35] R. Buyya. *High Performance Cluster Computing: Architectures and Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.
- [36] S. Byna, Xian-He Sun, W. Gropp, and R. Thakur. Predicting memory-access cost based on data-access patterns. In *CLUSTER '04: Proceedings of the 2004 IEEE International Conference on Cluster Computing*, pages 327–336, Washington, DC, USA, 2004. IEEE Computer Society.
- [37] C. Campbell. Constructive learning techniques for designing neural network systems. In *Neural Network Systems Technologies and Applications*, 1997.
- [38] P. Cao, E. W. Felten, A. R. Karlin, and K. Li. A study of integrated prefetching and caching strategies. In *Proceedings of the 1995 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 188–197. ACM Press, May 1995.
- [39] G. A. Carpenter. Neural network models for pattern recognition and associative memory. *Neural Networks*, 2:243–257, 1989.

- [40] G. A. Carpenter and S. Grossberg. Art 2: Selforganisation of stable category recognition codes for analog input patterns. *Applied Optics*, 26:4919–4930, 1987.
- [41] G. A. Carpenter and S. Grossberg. Art3: Hierarchical search using chemical transmitters in self-organizing pattern recognition architectures. *Neural Networks*, 3:129–152, 1990.
- [42] G. A. Carpenter, S. Grossberg, N. Markuzon, J.H. Reynolds, and D.B. Rosen. Fuzzy artmap: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, 3(5):698–713, September 1992.
- [43] G. A. Carpenter, S. Grossberg, and J.H. Reynolds. Artmap: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network. *Neural Networks*, 4:565–588, 1991.
- [44] G. A. Carpenter, S. Grossberg, and D.B. Rosen. Fuzzy art: Fast stable learning and categorization of analog patterns by an adaptive resonance system. *Neural Networks*, 4:759–771, 1991.
- [45] M. W. Chang, B. J. Chen, and C. J. Lin. Eunate network competition: Electricity load forecasting. citeseer.ist.psu.edu/677671.html.
- [46] G. J. Chappell and J. G. Taylor. The temporal kohonen map. *Neural Networks Journal*, 6(3):441–445, 1993.
- [47] G. Cheng and A. Zell. Externally growing cell structures for pattern classification. In H. Bothe and R. Rojas, editors, *Proceeding of the ICSC Symposia on Neural Computation (NC'2000) May 23-26, 2000 in Berlin, Germany*. ICSC Academic Press, 2000.
- [48] J. Cheng and R. Greiner. Learning Bayesian belief network classifiers: Algorithms and system. *Lecture Notes in Computer Science*, 2056:141–??, 2001.
- [49] K. S. Cho and S. Y. Lee. Implementation of infomax ica algorithm with analog cmos circuits. citeseer.ist.psu.edu/554958.html, 2001.
- [50] P. Chung. Checkpointing in cosmic: a user-level process migration environment, 1997.
- [51] M. Codish and C. Taboch. A semantic basis for termination analysis of logic programs and its realization using symbolic norm constraints. In *ALP/HOA*, pages 31–45, 1997.
- [52] S. Cohn, N. Sivakumaran, and R. Todling. A fixed-lag kalman smoother for retrospective data assimilation. In *Mon. Wea. Rev.*, 122, pages 2838–2867., 1994.
- [53] F. F. Computer. Fast, single-pass k-means algorithms. <http://citeseer.ist.psu.edu/327212.html>.
- [54] J. Corbalan, X. Martorell, and J. Labarta. Improving Gang Scheduling through Job Performance Analysis and Malleability. In *International Conference on Supercomputing*, pages 303–311, Sorrento, Italy, June 2001.

- [55] T. Cortes. *Cooperative Caching and Prefetching in Parallel/Distributed File Systems*. PhD thesis, UPC: Universitat Politècnica de Catalunya, Barcelona, Spain, 1997.
- [56] T. Cortes, S. Girona, and J. Labarta. PACA: A cooperative file system cache for parallel machines. Technical Report 96-07, UPC-CEPBA, 1996.
- [57] T. Cortes, S. Girona, and J. Labarta. Avoiding the cache-coherence problem in a parallel/distributed file system. Technical Report UPC-CEPBA-1996-13, UPC-CEPBA, May 1997.
- [58] T. Cortes, S. Girona, and J. Labarta. Design issues of a cooperative cache with no coherence problems. In *Proceedings of the Fifth Workshop on Input/Output in Parallel and Distributed Systems*, pages 37–46, San Jose, CA, November 1997. ACM Press.
- [59] T. Cortes and J. Labarta. Linear aggressive prefetching: A way to increase the performance of cooperative caches. In *Proceedings of the Joint International Parallel Processing Symposium and IEEE Symposium on Parallel and Distributed Processing*, pages 45–54, San Juan, Puerto Rico, April 1999.
- [60] Toni Cortes, Sergi Girona, and Jesús Labarta. Avoiding the cache-coherence problem in a parallel/distributed file system. In *Proceedings of High-Performance Computing and Networking*, pages 860–869, April 1997.
- [61] A. Dammindra, H. K. Saman, and S. Bala. Dynamic self-organizing maps with controlled growth for knowledge discovery. *IEEE Transactions on Neural Networks*, 11(3):601–614, May 2000.
- [62] S. P. Day and M. R. Davenport. Continuous-time temporal back-propagation with adaptable time delays. *IEEE Transactions on Neural Networks*, 4(2):348–354, March 1993.
- [63] J. Fernández de Canete, A. García-Cerezo, I. García-Moral, A. García-González, and C. Macías. Control architecture based on radial basis function network. application to a fluid level system. In *NICROSP '96: Proceedings of the 1996 International Workshop on Neural Networks for Identification, Control, Robotics, and Signal/Image Processing (NICROSP '96)*, page 254, Washington, DC, USA, 1996. IEEE Computer Society.
- [64] J. M. del Rosario and A. Choudhary. High performance I/O for parallel computers: Problems and prospects. *IEEE Computer*, 27(3):59–68, March 1994.
- [65] M. V. Devarakonda and R. K. Iyer. Predictability of Process Resource Usage: A Measurement-based Study on UNIX. *IEEE Transactions on Software Engineering*, 15(12):1579–1586, 1989.
- [66] M. Ding, Q. Zhou, and Q. Wang. The application of self-adaptive kalman filter in ngimu/gps integrated navigation system. In *ISDA '06: Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications (ISDA'06)*, pages 61–65, Washington, DC, USA, 2006. IEEE Computer Society.
- [67] G. Dobliger. An adaptive kalman filter for the enhancement of noisy ar signals, 1998.

- [68] E. Dodonov. Um mecanismo integrado de *Cache e Prefetching* para sistemas de entrada e saída de alto desempenho. Master's thesis, DC/UFSCar, February 2004.
- [69] E. Dodonov. Instrumentando e monitorando aplicativos. <http://www.dicas-l.com.br/dicas-l/20060427.php>, 2006.
- [70] E. Dodonov. Statmonitor project. <http://statmonitor.sourceforge.net/>, 2006.
- [71] E. Dodonov and R. F. de Mello. A model for automatic on-line process behavior extraction, classification and prediction in heterogeneous distributed systems. In *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pages 899–904, Washington, DC, USA, 2007. IEEE Computer Society.
- [72] E. Dodonov, R. Mello, and L. T. Yang. Adaptive technique for automatic communication access pattern discovery applied to data prefetching in distributed applications using neural networks and stochastic models. In *Proceedings of the ISPA-06*, November 2006.
- [73] E. Dodonov, R. F. Mello, and L. T. Yang. A network evaluation for lan, man and wan grid environments. In Laurence Tianruo Yang, Makoto Amamiya, Zhen Liu, Minyi Guo, and Franz J. Rammig, editors, *EUC*, volume 3824 of *Lecture Notes in Computer Science*, pages 1133–1146. Springer, 2005.
- [74] E. Dodonov, J. Q. Sousa, and H. C. Guardia. Gridbox: securing hosts from malicious and greedy applications. In *Proceedings of the 2nd workshop on Middleware for grid computing*, pages 17–22, New York, NY, USA, 2004. ACM Press.
- [75] Zümray Dokur and Tamer Ölmez. Segmentation of ultrasound images by using a hybrid neural network. *Pattern Recogn. Lett.*, 23(14):1825–1836, 2002.
- [76] D. Déharbe. A tutorial introduction to symbolic model checking. *Logic for concurrency and synchronisation*, 1:215–237, 2003.
- [77] A. L. Erkki. Time series prediction competition: The cats benchmark. cite-seer.ist.psu.edu/734570.html.
- [78] M. Erñanský, M. Makula, and U. Beňušková. Organization of the state space of a simple recurrent network before and after training on recursive linguistic structures. *Neural Networks.*, 20(2):236–244, 2007.
- [79] G. Evensen. An ensemble kalman smoother for nonlinear dynamics. In *Mon. Weather Rev.*, 128, pages 1852–1867, 2000.
- [80] R. Everson and S. Roberts. Non-stationary independent component analysis. In *Proceedings of ICANN-99*, 1999.
- [81] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532, Denver 1989, 1990. Morgan Kaufmann, San Mateo.

- [82] W. Fang, C. L. Wang, W. Zhu, and F. C. M. Lau. Pat: a postmortem object access pattern analysis and visualization tool. In *CCGRID 2004, 4th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2004), April 19-22, 2004, Chicago, Illinois, USA*, pages 379–386. IEEE Computer Society, 2004.
- [83] I. Farkas and L. Chud’y. Modified dynamic cell structures as a thinning algorithm. In *Proc. of 1-st Slovak Neural Network Symposium*, pages 71–80, November 1996.
- [84] U. M. Fayyad, C. Reina, and P. S. Bradley. Initialization of iterative refinement clustering algorithms. In *Knowledge Discovery and Data Mining*, pages 194–198, 1998.
- [85] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong. Theory and Practice in Parallel Job Scheduling. In *Job Scheduling Strategies for Parallel Processing*, volume 1291, pages 1–34. Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.
- [86] S. Fine, Y. Singer, and N. Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine Learning*, 32(1):41–62, 1998.
- [87] P. C. Fischer. On formalisms for turing machines. *J. ACM*, 12(4):570–580, 1965.
- [88] B. D. Fleisch and G. J. Popek. Mirage: A coherent distributed shared memory design. In *SOSP12*, pages 211–223, December 1989.
- [89] J. A. Freeman and D. M. Skapura. *Neural networks: algorithms, applications, and programming techniques*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1991.
- [90] J. A. Freeman and D. M. Skapura. *Neural Networks: Algorithms, Applications and Programming Techniques*. Addison-Wesley, 1991.
- [91] B. Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, Cambridge MA, 1995.
- [92] B. Fritzke. A self-organizing network that can follow non-stationary distributions. In *ICANN’97: International Conference on Artificial Neural Networks*, pages 613–618. Springer, 1997.
- [93] G. R. Ganger, B. L. Worthington, R. Y. Hou, and Y. N. Patt. Disk subsystem load balancing: Disk striping vs. conventional data placement. In *Proceedings of the Twenty-Sixth Annual Hawaii International Conference on System Sciences*, volume I, pages 40–49, 1993.
- [94] H. Gao, R. Sollacher, and H. P. Kriegel. Spiral recurrent neural network for online learning. In *Proceedings of 15th European Symposium on Artificial Neural Networks*, pages 483–488, 2007.
- [95] A. Felix Gers and Juergen Schmidhuber. Long short-term memory learns context free and context sensitive languages. In *IDSIA-03-00*, 25 2000.

- [96] F. Gers. Long short term memory in recurrent neural networks. cite-seer.ist.psu.edu/article/gers01long.html, 2001.
- [97] F. Gers, N. Schraudolph, and J. Schmidhuber. Learning precise timing with lstm recurrent networks. In *Journal of Machine Learning Research*, volume 3, pages 115–143, 2002.
- [98] F.A. Gers, F.A. Gers, and J. Schmidhuber. Recurrent nets that time and count. In J. Schmidhuber, editor, *Proc. IEEE-INNS-ENNS International Joint Conference on Neural Networks IJCNN 2000*, volume 3, pages 189–194 vol.3, 2000.
- [99] R. Gibbons. A Historical Application Profiler for Use by Parallel Schedulers. In *Job Scheduling Strategies for Parallel Processing*, pages 58–77. Springer Verlag, 1997.
- [100] F. Giunchiglia and P. Traverso. Planning as model checking. In *ECP*, pages 1–20, 1999.
- [101] R. Goularte, R. F. De Mello, E. Dodonov, and L. T. Yang. A model to estimate transcoding costs applied in live-video transmissions. *IPC*, 0:341–348, 2007.
- [102] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, September 1996.
- [103] W. D. Gropp and E. Lusk. *User's Guide for mpich, a Portable Implementation of MPI*. Mathematics and Computer Science Division, Argonne National Laboratory, 1996. ANL-96/6.
- [104] Hélio Crestana Guardia. *Considerações sobre as estratégias de um Sistema de Arquivos Paralelos integrado ao processamento distribuído*. PhD thesis, EPUSP, 1999.
- [105] H. Guo and W. Hsu. A survey on algorithms for real-time bayesian network inference. In *joint AAAI-02/KDD-02/UAI-02 workshop on Real-Time Decision Support and Diagnosis Systems*, 2002.
- [106] M. Hagiwara. Boltzmann machine learning and mean field theory learning with momentum terms. *Journal of Artificial Neural Networks*, 2(1-2):17–25, 1995.
- [107] F. Hamker and D. Heinke. Implementation and comparison of growing neural gas. In *Technical Report 1/97, Technical University of Ilmenau*, 1997.
- [108] J. Hammerton. Named entity recognition with long short-term memory. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003*, pages 172–175, Morristown, NJ, USA, 2003. Association for Computational Linguistics.
- [109] L. Hansen, J. Larsen, and T. Kolenda. On independent component analysis for multimedia signals. In *Multimedia Image and VideoProcessing, CRC Press, 2000*, 2000.
- [110] M. Harchol-Balter and A. B. Downey. Exploiting Process Lifetimes Distributions for Dynamic Load Balancing. *ACM Transactions on Computer Systems*, 15(3):253–285, August 1997.

- [111] P. Haslum. Model checking by random walk. In *Proceedings of ECSEL Workshop*, 1999.
- [112] Ji He, Ah-Hwee Tan, and Chew-Lim Tan. Modified art 2a growing network capable of generating a fixed number of nodes. *IEEE Transactions on Neural Networks*, 15(3):728–737, 2004.
- [113] D. O. Hebb. *The Organization of Behavior*. John Wiley, New York, 1949.
- [114] D. Heckerman. A tutorial on learning with bayesian networks. <http://citeseer.ist.psu.edu/heckerman96tutorial.html>, 1995.
- [115] M. Rauch Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *IEEE Symposium on Foundations of Computer Science*, pages 453–462, 1995.
- [116] L. J. Heyer, S. Kruglyak, and S. Yooseph. Exploring expression data: Identification and analysis of coexpressed genes. *Genome Res.*, 9(11):1106–1115, November 1999.
- [117] S. Hirose, K. Shimizu, S. Kanai, Yu. Kuroda, and T. Noguchi. Poodle-1: a two-level svm prediction system for reliably predicting long disordered regions. *Bioinformatics*, 23(16):2046–2053, 2007.
- [118] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- [119] P. Højén-Sørensen, L. Hansen, and O. Winther. Mean field implementation of bayesian ICA. In *In proceedings of 3rd International Conference on Independent Component Analysis and Blind Signal Separation (ICA2001)*. Institute of Neural Computation, 2001.
- [120] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Neurocomputing: foundations of research*, pages 457–464, 1988.
- [121] X. Hu, D. V. Prokhorov, and D. C. Wunsch II. Time series prediction with a weighted bidirectional multi-stream extended kalman filter. *Neurocomput.*, 70(13-15):2392–2399, 2007.
- [122] K. Huang, I. King, and M. Lyu. Constructing a large node chow-liu tree based on frequent itemsets. In *Proceedings of the International Conference on Neural Information Processing (ICONIP2002)*, 2002.
- [123] A. Hyvärinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE Transactions on Neural Networks*, 10(3):626–634, 1999.
- [124] Paragon XP/S product overview. Intel Corporation, 1991.
- [125] T. Jaditz. Time series prediction: Forecasting the future and understanding the past : Andreas s. weigend and neil a. gershenfeld, eds., (reading, ma: Addison-we. *Journal of Economic Behavior & Organization*, 26(2):302–305, March 1995. available at <http://ideas.repec.org/a/eee/jeborg/v26y1995i2p302-305.html>.
- [126] H. Jaeger. Echo state network. *Scholarpedia*, 2(9):2330, 2007.

- [127] H. Jahanian, G. A. Hossein-Zadeh, H. Soltanian-Zadeh, and B. A. Ardekani. Controlling the false positive rate in fuzzy clustering using randomization: application to fmri activation detection. *Magnetic Resonance Imaging*, 22(5):631–638, June 2004.
- [128] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley and Sons, 1991.
- [129] D. Jiang, C. Tang, and A. Zhang. Cluster analysis for gene expression data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1370–1386, 2004.
- [130] M. Kallahalla and P. J. Varman. Optimal prefetching and caching for parallel I/O systems. In *Proceedings of the Thirteenth Symposium on Parallel Algorithms and Architectures*, pages 219–228. ACM Press, July 2001.
- [131] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME—Journal of Basic Engineering*, 82(Series D):35–45, 1960.
- [132] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):881–892, 2002.
- [133] S. Kaski and E. Oja. *Kohonen Maps*. Elsevier Science Inc., New York, NY, USA, 1999.
- [134] P. Keleher, S. Dwarkadas, A. L. Cox, and W. Zwaenepoel. Treadmarks: Distributed shared memory on standard workstations and operating systems. In *Proc. of the Winter 1994 USENIX Conference*, pages 115–131, 1994.
- [135] D. Keskar and M. Leibowitz. Speeding up openoffice . profiling, tools, approaches. In *OpenOffice.org Conference*, 2005.
- [136] S. Kim, S. Imoto, and S. Miyano. Dynamic bayesian network and nonparametric regression for nonlinear modeling of gene networks from time series gene expression data. *Biosystems*, 75(1-3):57–65, July 2004.
- [137] S.-J. Kim and R.A. Iltis. Performance comparison of particle and extended kalman filter algorithms for gps c/a code tracking and interference rejection. cite-seer.ist.psu.edu/article/kim02performance.html.
- [138] T. Kimbrel and A. R. Karlin. Near-optimal parallel prefetching and caching. In *IEEE Symposium on Foundations of Computer Science*, pages 540–549, 1996.
- [139] T. Kimbrel, A. Tomkins, R. H. Patterson, B. Bershad, P. Cao, E. Felten, G. Gibson, A. R. Karlin, and K. Li. A trace-driven comparison of algorithms for parallel prefetching and caching. In *Proceedings of the 1996 Symposium on Operating Systems Design and Implementation*, pages 19–34. USENIX Association, October 1996.
- [140] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, Number 4598, 13 May 1983, 220, 4598:671–680, 1983.

- [141] S. Kirshner, P. Smyth, and A. W. Robertson. Conditional chow-liu tree structures for modeling discrete-valued vector time series. In *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 317–324, Arlington, Virginia, United States, 2004. AUAI Press.
- [142] M. Kohler. Using the kalman filter to track human interactive motion - modelling and initialization of the kalman filter for translational motion. Technical Report 629/1997, Fachbereich Informatik, Universität Dortmund, Fachbereich Informatik, Universität Dortmund, 44221 Dortmund, Germany, 1997.
- [143] T. Kohonen, S. Kaski, K. Lagus, J. Salojrvi, J. Honkela, V. Paatero, and A. Saarela. Self organization of a massive document collection. In *IEEE Transactions on Neural Networks*, 2000.
- [144] T. Koskela, M. Varsta, J. Heikkonen, and K. Kaski. Time series prediction using RSOM with local linear modesl. Technical Report B15, Helsinki University of Technology, Espoo, Finland, 1997.
- [145] D. Kotz. Bibliography about Parallel I/O. Available on the WWW at <http://www.cs.dartmouth.edu/pario/bib/>, 1994–2000.
- [146] D. Kotz and C. S. Ellis. Practical prefetching techniques for parallel file systems. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, pages 182–189. IEEE Computer Society Press, December 1991.
- [147] D. Kotz and C. S. Ellis. Caching and writeback policies in parallel file systems. *Journal of Parallel and Distributed Computing*, 17(1–2):140–145, January and February 1993.
- [148] D. Kotz and C. S. Ellis. Practical prefetching techniques for multiprocessor file systems. *Journal of Distributed and Parallel Databases*, 1(1):33–51, January 1993.
- [149] D. Kotz and C. S. Ellis. Practical prefetching techniques for multiprocessor file systems. In Hai Jin, Toni Cortes, and Rajkumar Buyya, editors, *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, chapter 17, pages 245–258. IEEE Computer Society Press and John Wiley & Sons, New York, NY, 2001.
- [150] D. Kotz and N. Nieuwejaar. Dynamic file-access characteristics of a production parallel scientific workload. In *Proceedings of Supercomputing '94*, pages 640–649, Washington, DC, November 1994. IEEE Computer Society Press.
- [151] D. Kotz and N. Nieuwejaar. Flexibility and performance of parallel file systems. *ACM Operating Systems Review*, 30(2):63–73, April 1996.
- [152] David Kotz. *Prefetching and Caching Techniques in File Systems for MIMD Multiprocessors*. PhD thesis, Duke University, April 1991. Available as technical report CS-1991-016.
- [153] S. C. Kremer. On the computational power of elman-style recurrent networks. *Neural Networks, IEEE Transactions on*, 6(4):1000–1004, 1995.

- [154] T. M. Kroegeer and D. D. E. Long. The case for efficient file access pattern modeling. In *Workshop on Hot Topics in Operating Systems*, pages 14–19, 1999.
- [155] B. Kröse and P. van der Smagt. *An introduction to neural networks*. URL ftp://ftp.informatik.uni-freiburg.de/papers/neuro/ann_intro_smag.ps.gz, The University of Amsterdam, 1996.
- [156] M. Kunze and J. Steffens. Growing cell structure and neural gas; incremental neural networks, 1995.
- [157] J. Lampinen and E. Oja. Self-organizing maps for spatial and temporal AR models. In Matti Pietikäinen and Juha Röning, editors, *Proc. 6 SCIA, Scand. Conf. on Image Analysis*, pages 120–127, Helsinki, Finland, 1989. Suomen Hahmontunnistustutkimuksen seura r. y.
- [158] C. R. F. Lara. Projeto de um servidor de vídeo sob demanda paralelo e distribuído. Master's thesis, DC/UFSCar, 2003.
- [159] A. Lendasse, M. Cottrell, V. Wertz, and M. Verleysen. Prediction of electric load using kohonen maps—application to the polish electricity consumption. In *Proceedings of the American control conference*, pages 3684–3689, 2002.
- [160] U. Lerner. Hybrid bayesian networks for reasoning about complex systems. In *PhD thesis, Stanford University*, October 2002.
- [161] J. M. Leski. An insensitive fuzzy c-means clustering. citeseer.ist.psu.edu/550525.html.
- [162] Z. Li. Computational design and nonlinear dynamics of a recurrent network model of the primary visual cortex. *Neural Computation*, 13(8):1749–1780, 2001.
- [163] T. W. Liao, Aivars K. Celmins, and II Robert J. Hammell. A fuzzy c-means variant for the generation of fuzzy term sets. *Fuzzy Sets Syst.*, 135(2):241–257, 2003.
- [164] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6(1):11–44, 1995.
- [165] Q. Ma and J. Wang. Application of bayesian neural networks to protein sequence classification. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 305–309, 2000.
- [166] P. Makhfi. Competitive learning. <http://www.makhfi.com/tutorial/clearn.htm>, 2006.
- [167] S. Marsland, J. Shapiro, and U. Nehmzow. A self-organising network that grows when required. *Neural Networks*, 15(8-9):1041–1058, 2002.
- [168] M. Martin, P. Harper, D. Sorin, M. Hill, and D. Wood. Using destination-set prediction to improve the latency /bandwidth tradeoff in shared memory multiprocessors. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, June 2003.

- [169] R. F. Mello. Proposta e avaliação de desempenho de um algoritmo de balanceamento de carga para ambientes distribuídos heterogêneos escaláveis. Tese (Doutorado) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Paulo, 2003.
- [170] R. F. Mello, J. A. Andrade, E. Dodonov, R. P. Ishii, and L. T. Yang. Optimizing distributed data access in grid environments by using artificial intelligence techniques. In Ivan Stojmenovic, Ruppia K. Thulasiram, Laurence Tianruo Yang, Weijia Jia, Minyi Guo, and Rodrigo Fernandes de Mello, editors, *ISPA*, volume 4742 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 2007.
- [171] R. F. Mello and L. J. Senger. A new migration model based on the evaluation of processes load and lifetime on heterogeneous computing environments. In *16th Symposium on Computer Architecture and High Performance Computing (SBAC'2004)*, pages 222–227, Foz do Iguaçu, PR, Brazil, 2004.
- [172] R. F. Mello, L. J. Senger, and L. T. Yang. Automatic text classification using an artificial neural network. *High Performance Computational Science and Engineering*, 1:1–21, 2005.
- [173] A. Meng. An introduction to markov and hidden markov models. <http://www2.imm.dtu.dk/pubdb/p.php?3313>, oct 2003.
- [174] C. D. Mitchell, R. A. Helzerman, L. H. Jamieson, and M. P. Harper. A parallel implementation of a hidden markov model with duration modeling for speech recognition. In *Digital Signal Processing, A Review Journal*, 5(1), pages 298–306, 1995.
- [175] F. Murtagh and A. Aussem. New problems and approaches related to large databases in astronomy. citeseer.ist.psu.edu/92142.html, 1998.
- [176] H. Mühlenbein and T. Mahnig. Evolutionary algorithms and the boltzmann distribution. citeseer.ist.psu.edu/730848.html.
- [177] M. Müller-Olm, D. A. Schmidt, and B. Steffen. Model-checking: A tutorial introduction. In *SAS '99: Proceedings of the 6th International Symposium on Static Analysis*, pages 330–354, London, UK, 1999. Springer-Verlag.
- [178] W. E. Nagel, A. Arnold, M. Weber, H. C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and analysis of MPI resources. *Supercomputer*, 12(1):69–80, 1996.
- [179] V. K. Naik, S. K. Setia, and M. S. Squillante. Processor Allocation in Multiprogrammed Distributed-memory Parallel Computer Systems. *Journal of Parallel and Distributed Computing*, 47(1):28–47, 1997.
- [180] S. Nakamori, R. Caballero-Águila, A. Hermoso-Carazo, J. Jiménez-López, and J. Linares-Pérez. Signal estimation algorithm from uncertain observations with signal and coloured noise correlated at any sampling time. In *MIC'06: Proceedings of the 25th IASTED international conference on Modeling, identification, and control*, pages 411–416, Anaheim, CA, USA, 2006. ACTA Press.

- [181] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. S. Ellis, and M. Best. File-access characteristics of parallel scientific workloads. Technical Report PCS-TR95-263, Dept. of Computer Science, Dartmouth College, August 1995.
- [182] F. Palacios-González. Tvcars models for forecasting. *Neurocomput.*, 70(13-15):2379–2391, 2007.
- [183] V. Pavlovic, B. Frey, and T. S. Huang. Time series classification using mixed-state dynamic bayesian networks. In *IEEE Conf. Computer Vision and Pattern Recognition*, Ft. Collins, CO, 1999.
- [184] J. A. Pérez-Ortiz, F. A. Gers, D. E., and J. Schmidhuber. Kalman filters improve LSTM network performance in problems unsolvable by traditional recurrent nets. In *Neural Networks*, volume 16(2), 2003.
- [185] D. Pham and J. Prince. An adaptive fuzzy c-means algorithm for image segmentation in the presence of intensity inhomogeneities. In *SPIE Medical Imaging 1998: Image Processing*, volume 3338, pages 555–563, 1998.
- [186] M. Piovoso and P. A. Laplante. Kalman filter recipes for real-time image processing. *Real-Time Imaging*, 9(6):433–439, 2003.
- [187] A. B. Quadrado. Partool: Uma ferramenta semi-automática de transposição de código fonte. Master's thesis, DC/UFSCar, 2003.
- [188] C. Raphael. Automatic segmentation of acoustic musical signals using hidden markov models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(4):360–370, 1999.
- [189] J. P. Rosca, D. Erdogmus, J. C. Príncipe, and S. Haykin, editors. *Independent Component Analysis and Blind Signal Separation, 6th International Conference, ICA 2006, Charleston, SC, USA, March 5-8, 2006, Proceedings*, volume 3889 of *Lecture Notes in Computer Science*. Springer, 2006.
- [190] G. Ruffini, L. Cucurull, A. Flores, and A. Rius. A pim-aided kalman filter for gps tomography of the ionospheric electron content. In *Physics and Chemistry of the Earth*, 1998.
- [191] P. Runkle, L. Carin, L. Couchman, T. J. Yoder, and J. A. Bucaro. Multiaspect target identification with wave-based matched pursuits and continuous hidden markov models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(12):1371–1378, 1999.
- [192] Neural N. Sabrine. Incremental construction of lstm recurrent neural network. <http://citeseer.ist.psu.edu/619652.html>.
- [193] M. Sakr, C. Giles, S. Levitan, B. Horne, M. Maggini, and D. Chiarulli. On-line prediction of multiprocessor memory access patterns. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1564–1569, 1996.
- [194] S. Särkkä, A. Vehtari, and J. Lampinen. Cats benchmark time series prediction by kalman smoother with cross-validated noise density. *Neurocomput.*, 70(13-15):2331–2341, 2007.

- [195] M. Schöttner, M. Wende, R. Göckelmann, T. Bindhammer, U. Schmid, and P. Schulthess. A gaming framework for a transactional dsm system. In *CCGRID 2003, 3rd IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, 12-15 May 2003, Tokyo, Japan, pages 502–509. IEEE Computer Society, 2003.
- [196] M. Schuster. Fast k-means vector quantizer for very large amounts of data. cite-seer.ist.psu.edu/9717.html.
- [197] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond (Adaptive Computation and Machine Learning)*. The MIT Press, December 2001.
- [198] L. J. Senger. An instance-based approach for predicting parallel application execution times. In *Proceedings of I2TS*, Sao Carlos, Sao Paulo, Brazil, December 2004.
- [199] L. J. Senger, R. F. Mello, M. J. Santana, and R. C. Santana. An on-line approach for classifying and extracting application behavior on linux. In Laurence T. Yang and Minyi Guo, editors, *High Performance Computing: Paradigm and Infrastructure*, chapter 20. John Wiley and Sons, 2005.
- [200] L. J. Senger, M. J. Santana, and R. H. C. Santana. Uma nova abordagem para a aquisição de conhecimento sobre o comportamento de aplicações paralelas na utilização de recursos. In *Proceedings of WORKCOMP*, pages 79–85, Outubro 2002.
- [201] L. J. Senger, M. J. Santana, and R. H. C. Santana. A new approach fo acquiring knowledge of resource usage in parallel applications. In *Proceedings of International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'2003)*, pages 607–614, 2003.
- [202] L. J. Senger, M. J. Santana, and R. H. C. Santana. Using runtime measurements and historical traces for acquiring knowledge in parallel applications. In *International Conference on Computational Science (ICCS) (to appear)*. Springer Lect. Notes Comput. Sci., 2004.
- [203] K. C. Sevcik. Characterizations of Parallelism in Applications and their use in Scheduling. *Performance Evaluation Review*, 17(1):171–180, May 1989.
- [204] P. Shah, J. F. Pâris, and A. Amer. dentifying stable file access patterns. In *Proceedings of the 21st IEEE/12th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2004)*, pages 230–240, 2004.
- [205] F. A. B. D. Silva and I. D. Scherson. Improving Parallel Job Scheduling Using Runtime Measurements. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 18–38. Springer Verlag, 2000. Lect. Notes Comput. Sci. vol. 1911.
- [206] M. Skubiszewski. A hardware emulator for binary neural networks. In *Proceedings of the International Neural Network Conference*, volume 2, pages 555–558, 1990.

- [207] M. Skubiszewski. An exact hardware implementation of the boltzmann machine. In *SPDP: 4th IEEE Symposium on Parallel and Distributed Processing*. ACM Special Interest Group on Computer Architecture (SIGARCH), and IEEE Computer Society, 1992.
- [208] P. Smith and N. C. Hutchinson. Heterogeneous process migration: The Tui system. *Software Practice and Experience*, 28(6):611–639, 1998.
- [209] W. Smith, I. T. Foster, and V. E. Taylor. Predicting Application Run Times Using Historical Information. In *JSSPP*, pages 122–142, 1998.
- [210] P. Sollich. Bayesian methods for support vector machines: Evidence and predictive class probabilities. *Machine Learning*, 46(1-3):21–52, 2002.
- [211] V. V. Soloviev. Prefetching in segmented disk cache for multi-disk systems. In *Proceedings of the Fourth Workshop on Input/Output in Parallel and Distributed Systems*, pages 69–82, Philadelphia, May 1996. ACM Press.
- [212] E. A. Spessoto. Um mecanismo de ajuste dinâmico para diminuição de latência e do tempo de resposta em sistemas de arquivos paralelo. Master's thesis, DC/UFSCar, 2003.
- [213] C. Squires and J. Shavlik. Experimental analysis of aspects of the cascade-correlation learning architecture. In *Machine learning research group working paper 91-1*, Computer Sciences Department, University of Wisconsin-Madison., 1991.
- [214] J. M. Squyres and A. Lumsdaine. A Component Architecture for LAM/MPI. In *Proceedings, 10th European PVM/MPI Users' Group Meeting*, number 2840 in Lecture Notes in Computer Science, pages 379–387, Venice, Italy, September / October 2003. Springer-Verlag.
- [215] J. Steffens and M. Kunze. Implementation of the supervised growing cell structure on the CNAPS neurocomputer. In F. Fogelman-Soulie and P. Gallinari, editors, *ICANN '95. International Conference on Artificial Neural Networks. Neuronimes '95 Scientific Conference*, volume 2, pages 51–6, Paris, France, 1995. EC2 & Cie.
- [216] M. Steijvers and P. Grunwald. A recurrent network that performs a contextsensitive prediction task. In *Proceedings of the 18th Annual Conference of the Cognitive Science Society*, 1996.
- [217] J. Sum, C. Leung, and L. Chan. Extended kalman filter in recurrent neural network training and pruning. In *Technical report CS-TR-96-05*, Department of Computer Science and Engineering, The Chinese University of Hong Kong, June 1996.
- [218] J. A. K. Suykens and J. Vandewalle, editors. *Nonlinear Modeling: Advanced Black-Box Techniques (The K. U. Leuven Time Series Prediction Competition)*, pages 241–251. Kluwer Academic Publishers, 1998.
- [219] N. A. Thacker, I. Abraham, and P. Courtney. Supervised learning extensions to the clam network. *Neural Networks.*, 10(2):315–326, 1997.
- [220] N. A. Thacker and J. E. W. Mayhew. Designing a layered network for context sensitive pattern classification. *Neural Networks.*, 3(3):291–299, 1990.

- [221] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. In Pat Langley, editor, *Proceedings of ICML-00, 17th International Conference on Machine Learning*, pages 999–1006, Stanford, US, 2000. Morgan Kaufmann Publishers, San Francisco, US.
- [222] G. K. Venayagamoorthy. 2007 special issue: Online design of an echo state network based wide area monitor for a multimachine power system. *Neural Networks*, 20(3):404–413, 2007.
- [223] D. Verstraten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout. Isolated word recognition with the liquid state machine: a case study. *Inf. Process. Lett.*, 95(6):521–528, 2005.
- [224] J. Vesanto. Using the SOM and local models in time-series prediction. In *Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland, June 4–6*, pages 209–214. Helsinki University of Technology, Neural Networks Research Centre, Espoo, Finland, 1997.
- [225] M. Vinther. Independent component analysis of evoked potentials in eeg. <http://www.logicnet.dk/reports/EEG/ICA.htm>, 2006.
- [226] N. A. Vlassis, A. Dimopoulos, and G. K. Papakonstantinou. The probabilistic growing cell structures algorithm. In *ICANN '97: Proceedings of the 7th International Conference on Artificial Neural Networks*, pages 649–654, London, UK, 1997. Springer-Verlag.
- [227] G. M. Voelker, E. J. Anderson, T. Kimbrel, M. J. Feeley, J. S. Chase, A. R. Karlin, and H. M. Levy. Implementing cooperative prefetching and caching in a globally-managed memory system. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, pages 33–43. ACM Press, June 1998.
- [228] P. L. Wadler. Comprehending monads. In *Proceedings of the 1990 ACM Conference on LISP and Functional Programming, Nice*, pages 61–78, New York, NY, 1990. ACM.
- [229] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang. Phoneme recognition using time delay neural networks. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37:328–339, 1989.
- [230] E. A. Wan and R. Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pages 153–158, 2000.
- [231] A. S. Weigend and N. A. Gershenfeld. Time series prediction: Forecasting the future and understanding the past. In A. S. Weigend and N. A. Gershenfeld, editors, *Santa Fe Institute Studies in the Sciences of Complexity, Proceedings of the NATO Advanced Research Workshop on Comparative Time Series Analysis, held in Santa Fe, New Mexico, May 14-17, 1992, Reading, MA: Addison-Wesley, [c]1994, edited by Weigend, Andreas S.; Gershenfeld, Neil A., 1994.*

- [232] Y. Xi. An analytical model for buffer hit rate prediction. In *MSc Thesis, Dept. of Computing and Information Science, Queen's University*, June 2001.
- [233] R. A. Yaffee and M. McGee. *Introduction to Time Series Analysis and Forecasting: With Applications of SAS and SPSS*. Academic Press, Inc., Orlando, FL, USA, 2000.
- [234] P. Yee and S. Haykin. A dynamic regularized radial basis function network for nonlinear. In *IEEE Transaction on Signal Processing*, volume 47, pages 2503–2521, 1998.
- [235] Y. Ishikawa, M. Matsuda, T. Kudoh, H. Tezuka, and S. Sekiguchi. The design of a latency-aware mpi communication library. In *SWOPP03*, 2003.
- [236] M.L. Yuan and M. Xie. An incremental representation of conceptual symbols using rce neural network. In *ICDL '02: Proceedings of the 2nd International Conference on Development and Learning*, page 102, Washington, DC, USA, 2002. IEEE Computer Society.
- [237] G. Udny Yule. On a method of investigating periodicities in disturbed series, with special reference to wolfer's sunspot numbers. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 226:267–298, 1927.
- [238] N. Zhang and D. Poole. A simple approach to bayesian network computations. In *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*, pages 171–178, 1994.
- [239] J. Zhao and J. Shawe-Taylor. A recurrent network with stochastic weights. In *Internal Report, Department of Computer Science, Royal Holloway University of London.*, 1994.
- [240] A. Zien, U. Brefeld, and T. Scheffer. Transductive support vector machines for structured variables. In Zoubin Ghahramani, editor, *Proceedings of the 24th International Conference on Machine Learning (ICML 2007)*, pages 1183–1190, Corvallis, Oregon, USA, June 2007. ACM Press.