UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DA COMPUTAÇÃO

JOREL SETTIN

Um Algoritmo de Aprendizado por Reforço Para Redes Neurais Utilizando Metaotimização Estatística

Trabalho de Diplomação.

Prof. Dr. Paulo Martins Engel
Orientador

Porto Alegre, dezembro de 2010.

AGRADECIMENTOS

Devo esse trabalho e minha jornada de aprendizado à minha família, que sempre me incentivou e apoiou o meu desejo de estudar.

Agradeço aos colegas que estiveram comigo durante minha vida acadêmica, especialmente ao Rodrigo, Fernando e Maurício, e a todos os amigos que estiveram juntos durante minha vida.

Realizo esse trabalho com esforço e dedicação aos professores, tão pacientes, que contribuíram com sua experiência para a formação do meu intelecto e caráter, bem como aos membros da universidade.

Devo minha autoconfiança aos meus colegas e amigos, que reavivaram minha vontade e capacidade de realizar uma grande obra, especialmente a Shankar Dev e Lakshmi.

Dedico os resultados desse trabalho somente para o bem e evolução da sociedade.

Finalmente, relembro de Deus, a Quem todas as causas e resultados pertencem.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

$\dot{x}$	derivada de uma variável qualquer *(x)* em relação ao tempo
∇	função gradiente
D	distância, segundo alguma métrica
$\bar{x}$	média de uma variável *(x)*
σ	desvio-padrão
ρ	parcela de uma população de uma distribuição
$N(\bar{u}, \sigma)$	distribuição normal com média $\bar{u}$ e desvio-padrão σ
$N_1 \cup N_2$	união dos elementos de duas distribuições normais
P	probabilidade de uma variável em uma distribuição
t	tempo
T	Temperatura em função do tempo no algoritmo de recozimento simulado
T_0	Temperatura inicial no algoritmo de recozimento simulado
$[a,b]$	faixa contínua de valores entre a e b
k	uma constante qualquer
$sgm(x)$	função sigmoide de x, expressa por $\frac{1}{1+e^x}$
NEAT	Neuroevolução de topologias aumentativas (NeuroEvolution of Augmenting Topologies)

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Esse trabalho analisa as principais técnicas de treinamento de redes neurais para problemas de aprendizado por reforço, e finalmente propõe um novo modelo utilizando suas melhores características, além de metaotimização baseada em amostragem estatística. O estudo tem por objetivos a obtenção de um método com alta taxa de sucesso, baixo número de simulações do problema, mínima necessidade de prévia especificação de parâmetros de treinamento, adaptabilidade a diversas classes de problemas, ter um comportamento determinístico para obter soluções com a mínima complexidade necessária, e de baixo custo computacional. Compara-se o desempenho do algoritmo proposto às técnicas utilizadas no atual estado da arte, em problemas de controle de sistemas físicos, e outros comumente utilizados como teste de desempenho.

Palavras-Chave: metaotimização estatística, aprendizado por reforço, treinamento de redes neurais, arquiteturas de redes neurais, recozimento simulado, algoritmos genéticos, algoritmos evolutivos, redes neurais cascateadas.

A Reinforcement Learning Algorithm for Neural Networks Using Statistical Meta Optimization

ABSTRACT

This work analyzes the main techniques for training neural networks in reinforcement learning problems, and finally proposes a new algorithm using their best features, and besides, utilizing meta-optimization based on statistical sampling. The study aims to obtain a method with a high rate of success, low number of the simulated problem evaluations, minimal needs of prior training parameters specification, adaptability to different problem classes, deterministic behavior for obtaining minimal complexity solutions, and low computational cost. The performance of the new algorithm is compared to the main techniques used in state of the art, in problems of physical system control and other usual benchmarks.

Keywords: statistical meta-optimization, reinforcement learning, neural network training, neural network architecture, simulated annealing, genetic algorithms, evolutionary algorithms, cascaded neural networks.

1 INTRODUÇÃO

Técnicas de aprendizado por reforço são utilizadas quando se deseja omitir o trabalho de desenvolver a solução de um problema, porém devendo especificar uma métrica de qualidade de uma solução qualquer para o problema, normalmente uma medida quantitativa chamada escore. Desse modo cria-se uma função que avalia os resultados das ações de um ator em um ambiente de simulação, e utiliza-se o poder computacional para heuristicamente encontrar uma solução aceitável baseando-se na avaliação de diversas soluções. Usualmente uma solução pode ser representada por uma rede neural, que controla as ações do ator.

Problemas complexos caracterizam-se por possuírem soluções com estratégias diferentes que podem atingir a qualidade desejada; onde a qualidade das soluções cresce variavelmente com a complexidade da rede neural; onde a qualidade de soluções é dificilmente avaliada e pouco representativa da proximidade da função ótima; e onde até mesmo se desconhece se o problema possui solução ou qual é a solução ótima, além de possuir subsoluções não ótimas que devem ser evitadas. Essas situações têm como exemplo a natureza, onde são encontradas diversas soluções muitas vezes inspiradoras para a área da robótica; a complexidade algorítmica de estratégias de jogos, onde a complexidade computacional exigida para a obtenção de uma estratégia inicial é desconhecida; sistemas físicos caóticos, como a aerodinâmica; e mineração de dados.

Um dos principais objetivos dos métodos heurísticos para a busca de uma solução é que o algoritmo deve ser capaz de transpor ótimos locais, que são subsoluções de baixa qualidade, sendo capaz de alcançar soluções que cumpram o objetivo especificado, ou alcançando o ótimo global, que representa a melhor solução para um problema.

As duas principais abordagens para os algoritmos heurísticos de busca de soluções em problemas de aprendizado por reforço utilizando redes neurais são: as baseadas em algoritmos genéticos, que mantém uma população de redes neurais dividida em espécies, sofrendo mutações e recombinações ao longo do tempo, e sendo reavaliadas; e a dos métodos de recozimento simulado (*Simulated Annealing*), que iniciam com uma taxa de variação máxima para a rede neural, e ao longo do tempo diminuem a taxa de variação, desse modo convergindo para uma solução ótima.

No atual estado da arte têm sido propostas várias arquiteturas de redes neurais e variações e combinações dos algoritmos de aprendizagem, conseguindo-se bons resultados para aplicações específicas. Porém, ainda há poucos estudos que unifiquem a teoria de aprendizado de forma prática, e que descrevam porque cada heurística obtém um desempenho melhor em cada classe de problemas. Desse modo, ainda é tarefa do desenvolvedor decidir qual algoritmo utilizar e ajustar diversos parâmetros do aprendizado, o que pode ser uma tarefa exaustiva.

Neste trabalho objetiva-se analisar as principais heurísticas utilizadas atualmente, tentando fundamentar matematicamente a grande flexibilidade dos algoritmos genéticos, e analisando a complexidade computacional e desempenho nem sempre aceitáveis, porém mais precisos, dos métodos de recozimento simulado.

1.1 Aprendizado por Reforço

O problema de aprendizado por reforço foi introduzido para que sistemas computacionais pudessem controlar sistemas físicos (SUTTON, 1983), que contém elementos complexos obedecendo às leis matemáticas. Como se desconhecia a solução matemática exata para o problema, foi tentado utilizar o poder computacional para achar uma solução aceitável que cumprisse um determinado objetivo, sendo determinada apenas uma função que avalia o escore de uma solução.

Embora o conceito de aprendizado por reforço seja amplo, nesse trabalho o estudo será restringido ao uso de redes neurais para a obtenção de soluções, bem como o aprendizado por reforço será restringido ao sistema não supervisionado, significando que o treino de redes neurais não visa corrigir ações individuais da rede neural no sistema físico, mas avalia somente o resultado final (escore) de uma rede neural na simulação do problema a ser resolvido.

O algoritmo deve possuir sensores, ou entradas, que obtém uma informação do atual estado do sistema físico simulado, e, além disso, deve possuir atuadores, ou saídas, que realizam uma determinada ação no meio simulado.

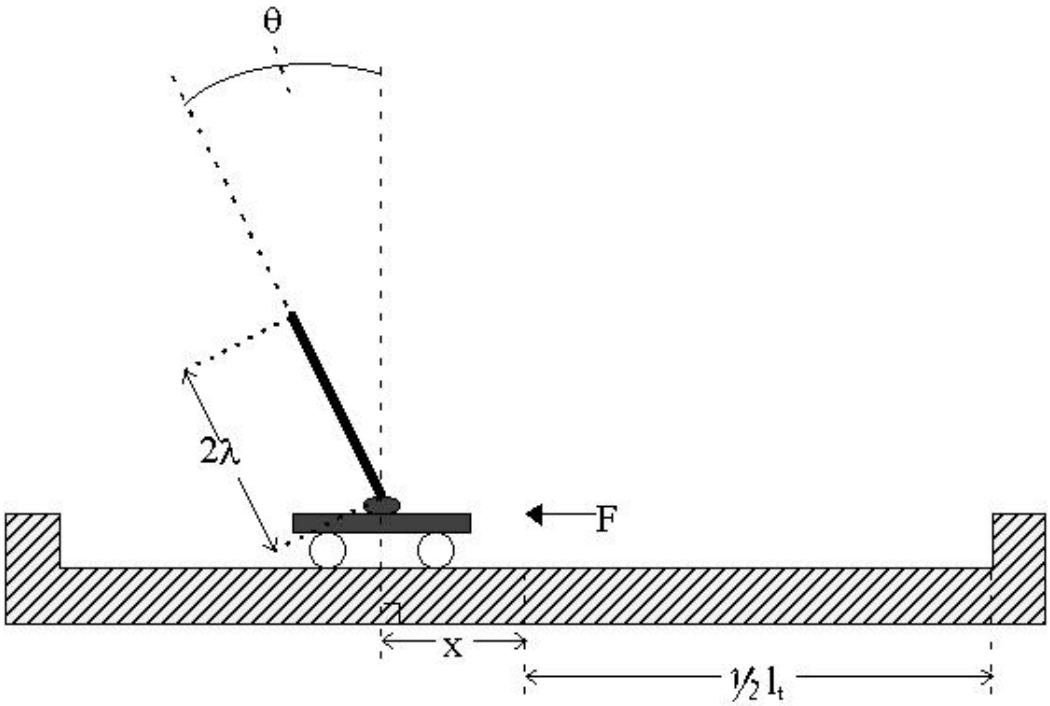


Figura 1.1: Problema do carro-pêndulo

(cart-pole)

A figura 1.1 ilustra um problema típico de controle, onde um carro pode mover-se horizontalmente no eixo *X*, através da aplicação de uma força horizontal *F*, devendo

manter o pêndulo equilibrado em relação à vertical. A solução desse problema pode ser calculada matematicamente utilizando como variáveis a posição do carro (*X),* a velocidade do carro $(\dot{X})$, a posição angular do pêndulo *(θ)* e a velocidade angular do pêndulo $(\dot{\theta})$. A medida de qualidade de uma solução é o tempo em que o sistema consegue manter o pêndulo equilibrado sem o carro ultrapassar um limite de posição (l_t).

A figura 1.2 apresenta uma arquitetura típica de rede neural, onde neurônios de entrada (a_n) recebem valores dos sensores do sistema, os neurônios ocultos (b_n) calculam a função de transferência do valor da soma das entradas multiplicadas por seus respectivos pesos (v_{np}), e os neurônios de saída (c_n) calculam a função de transferência do valor da soma dos neurônios ocultos multiplicados por seus respectivos pesos (w_{np}), e finalmente obtêm um valor de saída que será utilizado por um atuador no sistema simulado.

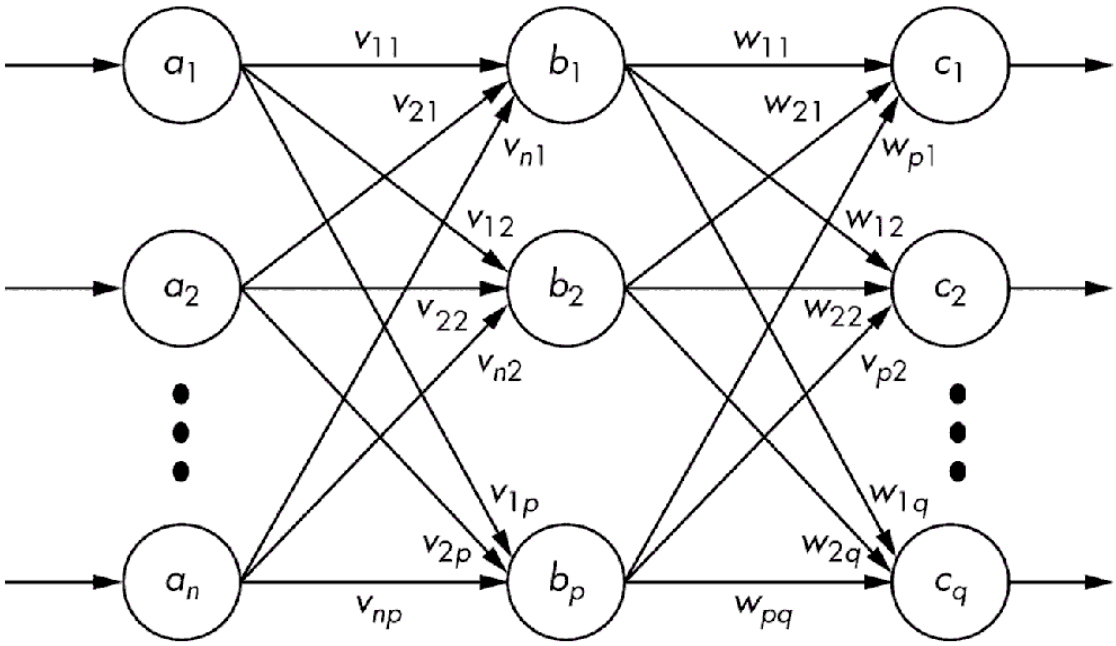


Figura 1.2: Rede neural multicamada

(feed-forward multilayer neural network)

A figura 1.3 ilustra o modelo de aprendizado por reforço supervisionado, onde um ator exerce uma ação sobre o ambiente, e o ambiente fornece variáveis de entrada *(estado)* para o ator. O crítico é capaz de alterar o funcionamento do ator segundo uma *política*, buscando melhorar seu desempenho, de acordo com a recompensa que é proporcional a qualidade da solução do ator. O método da figura difere brevemente do método utilizado nesse trabalho, que é um método não supervisionado. Nos métodos não supervisionados o crítico não conhece o estado atual do sistema, nem analisa as ações individuais do ator, assim leva em conta somente a recompensa oferecida pelo ambiente, e fornece algum parâmetro (por exemplo: um fator de mutação nos métodos genéticos, ou a temperatura nos métodos de recozimento simulado) ao ator para a realização do aprendizado. Assim, o problema simulado é desconhecido ao método de treinamento (BLACK Box, 2010).

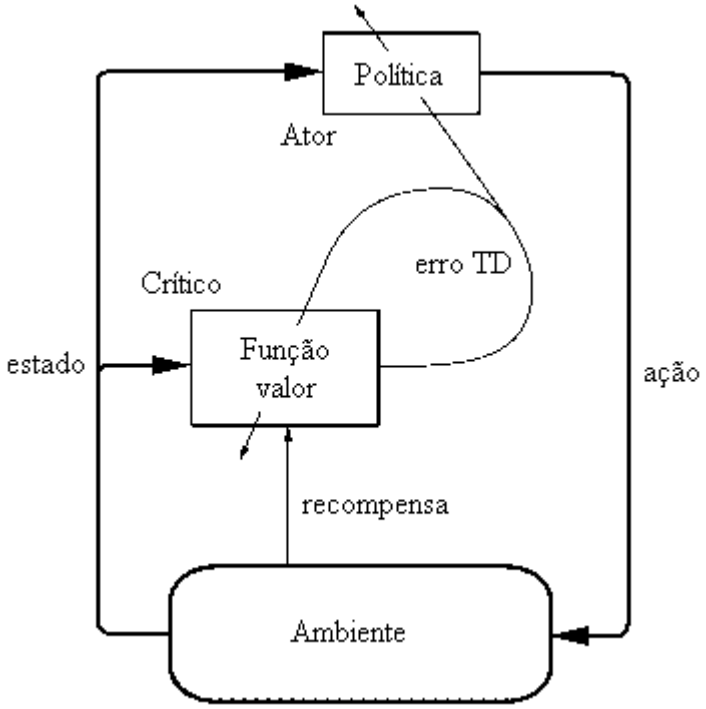


Figura 1.3: Modelo de aprendizado por reforço

(SUTTON, 1983)

1.2 Principais Métodos de Aprendizado Aplicáveis ao Problema

1.2.1 Métodos Genéticos

Os métodos de aprendizado genéticos baseiam-se fortemente na evolução biológica dos seres vivos. Cada neurônio e conexão de uma rede neural podem ser representados por genes, portanto a rede neural é representada por um conjunto de genes ou genoma. Usualmente é mantida uma especiação, ou seja, os genomas mais parecidos, segundo alguma métrica, são agrupados em espécies. Ao longo de gerações, ocorrem mutações nos genes, representando mutações nos pesos da rede neural, ou são criados novos genes, representando a criação de novos neurônios e conexões em uma rede neural. Também é realizada a recombinação genética entre indivíduos da mesma espécie, na esperança de aproveitar as melhores características de dois genomas.

Existem diversas variações dos algoritmos genéticos, contendo diferentes representações dos genótipos e fenótipos, diferentes técnicas de recombinação genética e de mutações, e diferentes técnicas de seleção das melhores soluções.

Normalmente a utilização de algoritmos genéticos exige a escolha de parâmetros como o tamanho populacional, para garantir uma grande variabilidade de soluções iniciais; o grau de especiação, para manter a variabilidade de nichos de soluções; o grau de mutação; o grau de eliminação de espécies que não apresentam melhora de desempenho, para evitar ótimos locais; e o grau de seleção das melhores soluções para realizar a recombinação genética. Pode ser necessário um grande número de tentativas até escolherem-se parâmetros ótimos para obter uma solução aceitável de um determinado problema.

1.2.2 Recozimento Simulado

Os métodos de recozimento simulado de aprendizado baseiam-se na termodinâmica, especificamente na ideia de recozimento da siderurgia, onde aquecendo um material a uma alta temperatura faz com que os átomos sejam capazes de desprenderem-se de suas posições iniciais, e então, lentamente resfriando o material, os átomos tendem a entrar em um estado ordenado de baixa energia. Analogamente, no sistema de aprendizado chamado recozimento simulado *(Simulated Annealing)* inicia-se com uma grande taxa de variação (análogo a uma alta temperatura), causando uma grande variabilidade de soluções, e gradualmente reduz-se a taxa de variação (análogo ao resfriamento), porém utilizando uma taxa de aceitação que dá preferência a estados onde a solução possui maior desempenho (análogo ao fato dos átomos tenderem a estados de baixa energia). Desse modo, conforme a temperatura, ou taxa de variação, é baixada, é obtido o refinamento de uma solução até tender ao ótimo global. Normalmente é apresentada uma prova matemática de convergência ao ótimo global, assumindo distribuições de probabilidades de estados similares ao processo termodinâmico.

Os primeiros algoritmos de recozimento simulado eram fortemente baseados na termodinâmica, inclusive utilizando as mesmas equações de distribuição de probabilidades do espaço de busca. O método original descrito deve utilizar um roteiro de resfriamento muito lento com $T = T_0 / \log(t)$, sendo T a temperatura no instante t, além de iniciar com uma temperatura alta suficiente para sair de qualquer ótimo local presente no espaço de busca, o que leva a um tempo de computação excessivamente alto para objetivos práticos. Além disso, o espaço de soluções percorrido pelo algoritmo é muito superior ao próprio tamanho do espaço de soluções, pois o algoritmo mantém apenas um estado atual, podendo regressar ao mesmo aleatoriamente. Devido a esses problemas têm sido propostas diversas formas de acelerar o processo. Para evitar o contratempo de percorrer o espaço de soluções aleatoriamente, uma das soluções é iniciar em vários pontos do espaço de busca, assim sendo possível iniciar com uma temperatura menor e convergir em menos tempo. Para aumentar a probabilidade de obter bons candidatos a próximos estados, é possível utilizar alguma heurística sobre os melhores candidatos já encontrados, como analisar os pesos e *bias* das soluções (INGBER, 1989). O roteiro do resfriamento e a taxa de aceitação de candidatos estão intrinsecamente ligados ao gerador de candidatos, portanto muitos estudos propõem ambos relacionadamente.

De uma forma geral, os principais parâmetros a ser escolhidos para o algoritmo são: a energia (qualidade da solução) alvo; o gerador de candidatos; a probabilidade de aceitação, pois o algoritmo deve aceitar estados piores do que o atual com alguma probabilidade a fim de transpor ótimos locais e percorrer o espaço de soluções; a temperatura inicial; e o roteiro de resfriamento de temperatura.

1.2.3 Método do Gradiente Ascendente

O método do gradiente ascendente *(gradient ascent)* baseia-se na ideia de que é possível derivar a qualidade de uma solução em relação aos pesos e *bias* de uma rede neural, obtendo o gradiente (∇) do escore e assim variando-os em direção ao máximo escore.

A figura 1.4 ilustra quatro passos sucessivos (x_n) na busca do ótimo global do espaço de busca (pequeno círculo central).

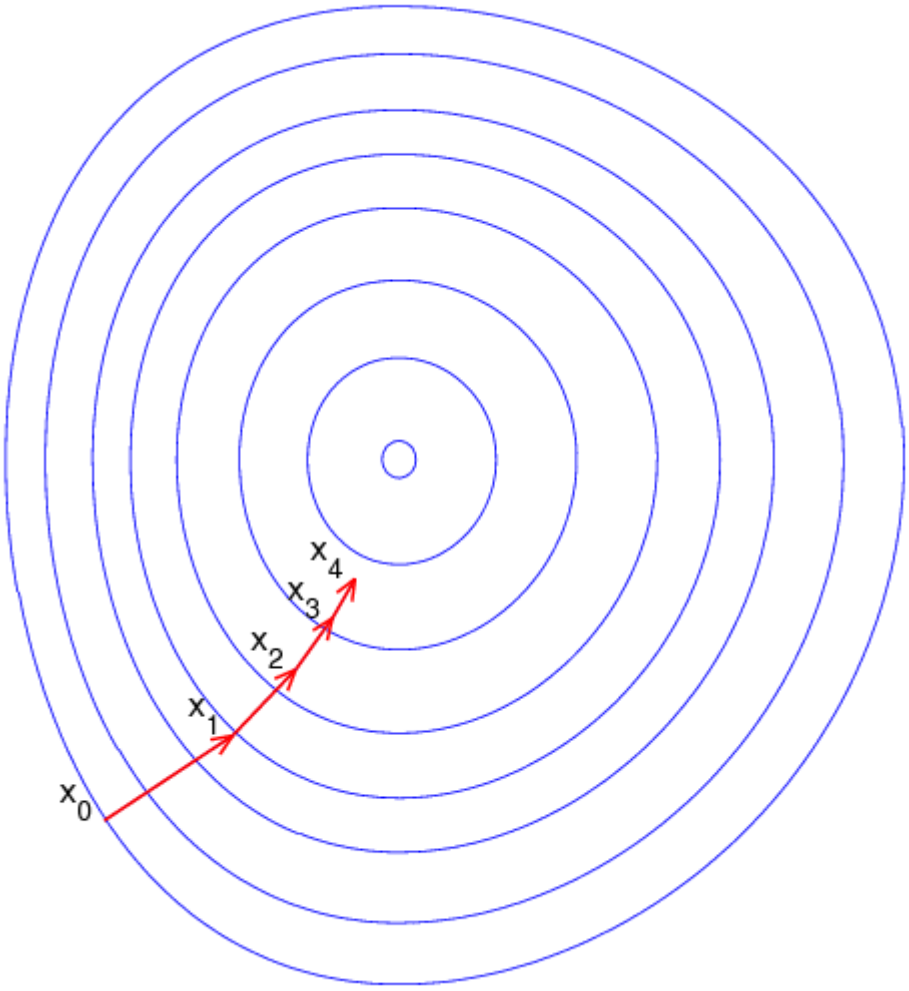


Figura 1.4: Método do gradiente ascendente

O método do gradiente ascendente exige que o espaço de busca seja convexo. Portanto, raramente funciona em um ambiente de aprendizado por reforço, embora seja presumível que haja algum grau de convexidade local nos problemas matemáticos e possa ser utilizado para um ajuste final dos pesos e *bias* da rede neural.

Foram desenvolvidos vários métodos matemáticos para o algoritmo ser capaz de transpor ótimos locais. Nesse trabalho não será abordado em detalhes esse método, porém sim algumas técnicas que foram desenvolvidas em decorrência das dificuldades desse método, na seção 2.3.

2 ANÁLISE DO ESTADO DA ARTE

Nessa seção são descritos dois dos mais eficientes métodos utilizados para otimização de problemas de aprendizado por reforço, sendo o primeiro genético e o segundo de recozimento simulado. Em seguida, na seção 2.3 são apresentados os métodos de aprendizado de redes neurais cascateadas, por conterem um princípio determinístico eficiente na busca de soluções a partir da complexidade mínima.

2.1 Neuroevolução de Topologias Aumentativas

Um dos métodos mais promissores, entre os que clamam ser o melhor método genético, está o algoritmo de neuroevolução de topologias aumentativas *(NeuroEvolution of Augmenting Topologies – NEAT).* Esse método reuniu um conjunto de técnicas e obteve o melhor resultado entre diversos algoritmos genéticos, especialmente em tarefas de controle de jogos (STANLEY, 2005).

2.1.1 Descrição

O método *NEAT* se baseia nos princípios de recombinação e mutação genética de indivíduos de uma população candidatos à solução do problema. Os genes (genótipo) contém uma representação de uma rede neural (fenótipo). Através das gerações, acontecem recombinações de genes e mutações, sendo esperado que uma solução melhor seja encontrada.

Os principais motivos pelo qual o método *NEAT* tem se mostrado promissor em buscas de estratégias complexas são: a manutenção de um número de inovação para cada novo gene, a fim de recombinar genes relacionados entre si; a especiação baseada em uma medida de diferença entre os genomas, calculada através do número de genes disjuntos (presentes em somente uma rede neural), o número de genes excessivos (presentes após o último gene relacionado), e a média da diferença de pesos entre os genes relacionados; a criação de uma população inicial com quantidade mínima de genes, aplicando o princípio de que é mais fácil treinar uma rede neural de dimensão pequena; e a manutenção de uma quantidade razoável de espécies, dando oportunidade de evolução em vários nichos de soluções.

Segue a explicação da figura 2.1 obtida do trabalho original, ilustrando o processo de recombinação genética do *NEAT*: “Embora o *Pai 1* e o *Pai 2* pareçam diferentes, seus números de inovação (mostrados na parte superior de cada gene) nos mostra quais genes estão relacionados entre si. Mesmo sem qualquer análise topológica, uma nova estrutura pode ser criada, combinando as interseções dos dois pais bem como suas partes diferentes. Genes relacionados são herdados aleatoriamente, enquanto genes disjuntos

(aqueles não relacionados, mostrados entre o primeiro e o último gene relacionado) e genes em excesso (aqueles não relacionados, mostrados após o último gene relacionado) são herdados do *Pai* mais adaptado. Nesse caso, igual adaptação é assumida, então os genes disjuntos e os genes em excesso são também herdados aleatoriamente. Os genes desabilitados podem se tornar habilitados novamente em futuras gerações: existe uma chance pré-determinada que um gene herdado esteja desabilitado se ele estiver desabilitado no pai também." (STANLEY, 2001). Nesse caso, "adaptação" refere-se ao escore de uma rede neural.

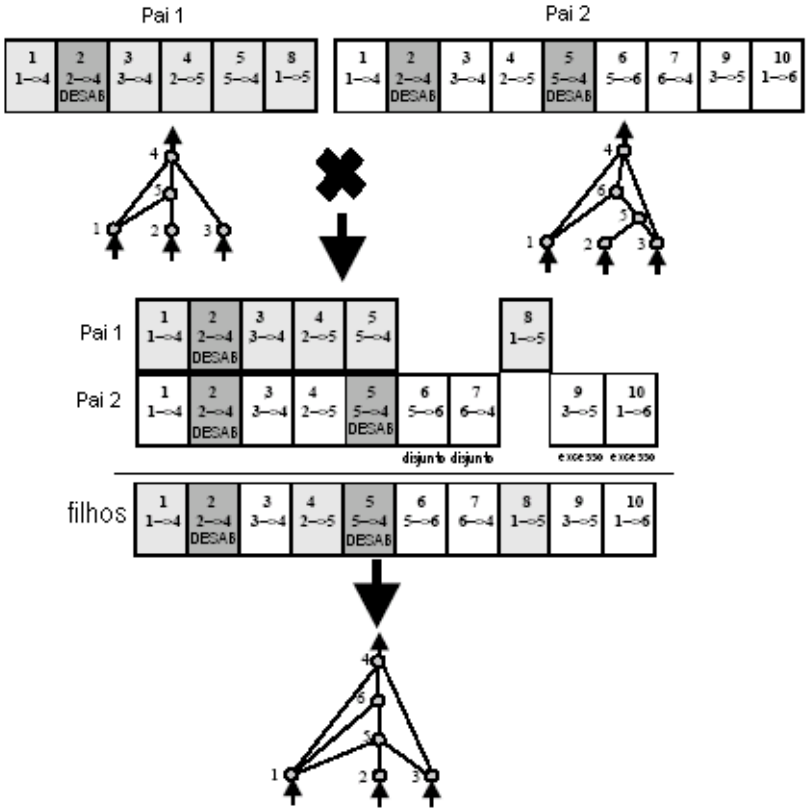


Figura 2.1: Casando genomas de diferentes topologias no NEAT

(STANLEY, 2001)

Segue a explicação original da figura 2.2 ilustrando as possíveis mutações do método *NEAT*. "Ambos os tipos de mutação, adicionando uma conexão e adicionando um nodo, são ilustrados com os genes conectores de uma rede neural, mostrados acima de seus fenótipos. O número superior ao genoma é o número da inovação daquele gene. Os números de inovações são marcadores históricos que identificam o ancestral histórico original de cada gene. Para os novos genes criados o número é incrementado. Adicionando uma conexão, um simples novo gene de conexão é adicionado ao fim do genoma dado o próximo número da inovação disponível. Adicionando um nodo, o gene de conexão sendo dividido é desabilitado, e dois novos genes de conexão são adicionados ao fim do genoma. O novo nodo fica entre as duas novas conexões. Um novo gene de nodo é adicionado ao genoma." (STANLEY, 2001).

O *NEAT* utiliza um tamanho fixo para população, e para manter um número mínimo de espécies, o escore de cada indivíduo é dividido pelo número de indivíduos na mesma espécie. Isso evita que uma espécie que obteve rapidamente um alto desempenho cresça demasiadamente e impeça a progressão de outras espécies, candidatas ao objetivo.

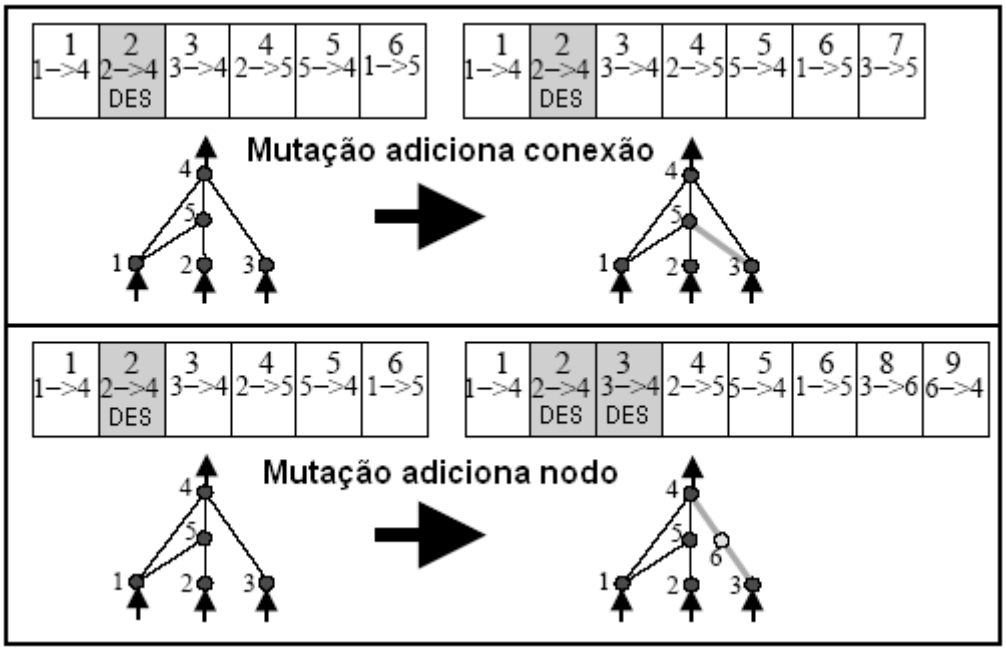
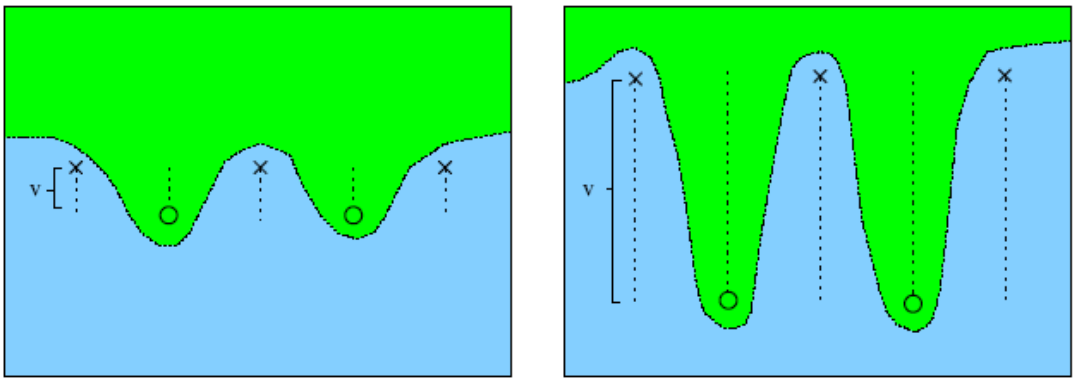


Figura 2.2: Os dois tipos de mutação estrutural no *NEAT*

(STANLEY, 2001)

2.1.2 Avaliação do Desempenho do Método *NEAT*

O algoritmo *NEAT* se mostrou eficiente para topologias relativamente pequenas, sendo suficiente para criar uma estratégia iterativa em jogos. Porém, um estudo apresentado a seguir ilustra as limitações do algoritmo quando é exigida uma topologia mais complexa e que exige um alto grau de variabilidade das soluções, demonstrando sua baixa eficiência nesses ambientes (KOHL, 2009).



(a) (b)

Figura 2.3: Problema de classificação de N-Pontos

(KOHL, 2009)

A figura 2.3 ilustra o problema da classificação de pontos em duas situações, uma com baixa variabilidade *(v)* em *(a)*, outra com alta variabilidade em *(b)*.

A figura 2.4 ilustra os resultados do desempenho do algoritmo *NEAT* para os diferentes graus de variabilidade exigidos. "Conforme *v* aumenta, a variabilidade necessária para resolver o problema também aumenta, e o desempenho do *NEAT* cai. O

mesmo ocorre conforme *N* aumenta. Esse resultado realça a dificuldade do *NEAT* em produzir soluções com alta variabilidade.” (KOHL, 2009).

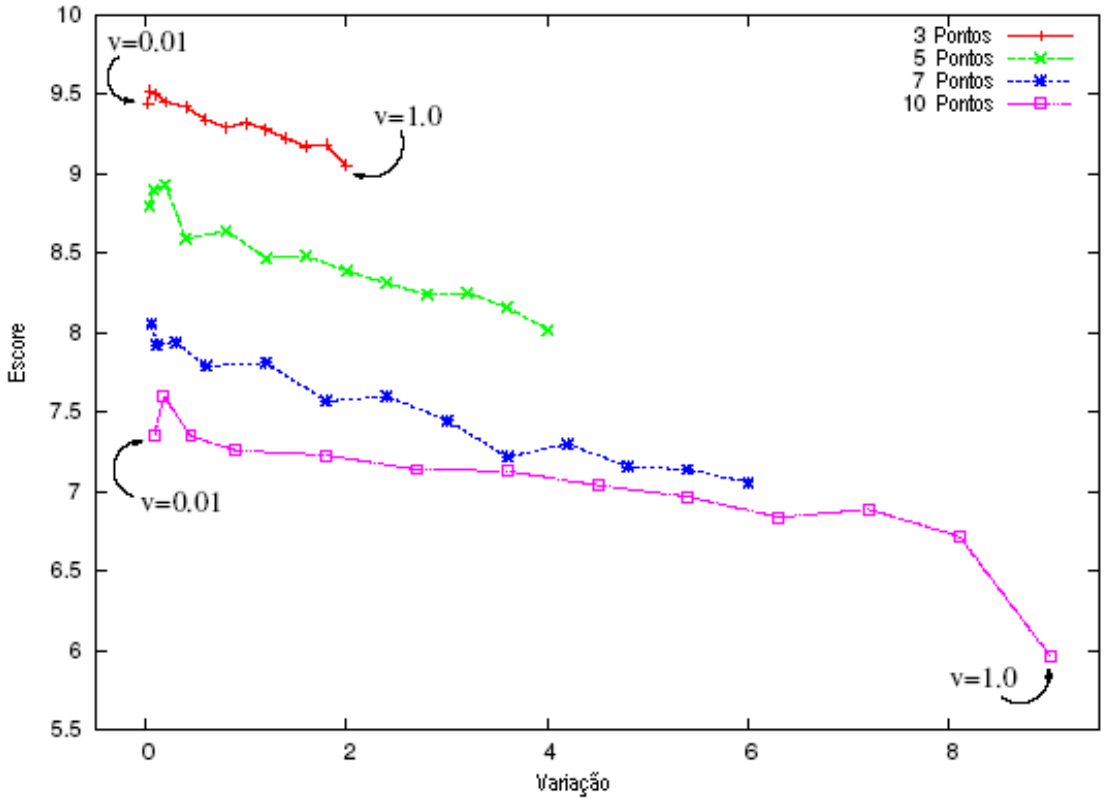


Figura 2.4: Desempenho do *NEAT* no problema de N-Pontos (para $N = 3, 5, 7, 10$ e $v = 0.01..10$). (KOHL, 2009).

Kohl (2009) analisa os motivos do baixo desempenho do algoritmo *NEAT*, chegando à conclusão que redes neurais com um número maior de neurônios haviam obtido melhores resultados. Assim, Kohl propõe aumentar a taxa de mutações estruturais do algoritmo de aprendizado, porém novamente obtendo resultados insatisfatórios. Chega-se a conclusão que tais tipos de problema necessitam a construção gradual e cuidadosamente controlada da rede neural. Finalmente Kohl propõe uma arquitetura *Cascade-NEAT*, onde novos neurônios são adicionados exatamente com os mesmos tipos de conexões apresentados no método *Cascade-Correlation Neural Network,* ilustrado na figura 2.5, obtendo soluções ligeiramente melhores do que o algoritmo original para o problema de N-Pontos, porém obtendo soluções piores para o problema de carro-pêndulo. Uma explicação do motivo do baixo desempenho no problema de carro-pêndulo é que esse problema exige apenas um neurônio para obtenção da solução, conforme é comentado na seção 4.1.

2.2 Recozimento Simulado Adaptativo

O método de recozimento simulado adaptativo *(Adaptive Simulated Annealing)* desenvolvido por Ingber (1989), previamente conhecido como *Very Fast Simulated Reannealing*, foi uma melhora matemática na fórmula para geração de candidatos do algoritmo de recozimento simulado que obteve uma aceleração significativa do algoritmo original, permitindo um roteiro de resfriamento com taxas exponenciais, e

assim tornando o algoritmo prático para diversos tipos de problemas, porém nem sempre alcançando o ótimo global.

2.2.1 Descrição

A versão anterior, chamada *Fast Simulated Annealing*, de H. Szu e R. Hartley, utiliza uma distribuição estatística de Cauchy (similar a uma distribuição normal, podendo assumir valores infinitos) para a geração de pesos *e bias* da rede neural, requerendo um roteiro para a queda de temperatura proporcional a $T = T_0 / t$ para obtenção do ótimo global. Finalmente Ingber propôs que os pesos e *bias* a ser otimizados encontram-se em uma faixa finita de valores $[A_i, B_i]$, e que esta faixa poderia iniciar com valores fixos de *A* e *B* e ser amostrada e estimada durante a execução do algoritmo de treinamento. Isso permitiu que o planejamento de resfriamento da temperatura fosse $T = T_0 / e^{(k \cdot t^{\frac{1}{D}})}$ sendo *k* uma constante a ser estimada pelo algoritmo de aprendizado ou empiricamente; *D* o número de dimensões a ser otimizadas (pesos e *bias* da rede neural); e *t* o tempo de simulação do algoritmo de aprendizado, permitindo assim a obtenção do ótimo global em um tempo prático.

Foram utilizadas várias técnicas para amostragem dos valores dos pesos e *bias* e assim obter um gerador de candidatos eficiente: o simples reescalonamento para uma distribuição uniforme $[A_i, B_i]$; a análise de covariância entre todos os valores (INGBER, 1992); e o cálculo da sensitividade da qualidade da solução em relação a cada parâmetro a ser otimizado.

2.3 Aprendizado de Redes Neurais em Cascata

No contexto da época do estudo (FAHLMAN, 1991), um dos principais problemas de utilizar redes neurais de topologia fixa, que iniciam com um número fixo de neurônios e não são criados novos, é que usualmente era causado um sobreajuste *(over-fitting)* de pesos e *bias* da rede neural. Isso significa que o algoritmo de aprendizado tentava ajustar-se a um ótimo local, devido ao desconhecimento de quais conjuntos de teste eram mais significativos para a obtenção de uma solução de boa qualidade em relação a todo o conjunto de teste. Além disso, o cálculo do gradiente ascendente para vários neurônios pode causar que os passos em direção ao ótimo global acabam não convergindo na direção correta, e desse modo deve-se reduzir o tamanho do passo a valores ineficientemente pequenos.

Fahlman (1991) propôs a arquitetura de redes neurais chamada Arquitetura de Aprendizado de Correlação Cascateada *(The Cascade-Correlation Learning Architecture)*, onde a rede neural inicia sem nenhum neurônio oculto, e somente um neurônio é adicionando e treinado a cada iteração, conforme a figura 2.5, assim obtendo automaticamente um tamanho mínimo para a rede neural, e evitando o sobreajuste por identificar as entradas mais relevantes para o desempenho da rede neural.

O método de aprendizado cascateado foi novamente melhorado, permitindo identificar quais as entradas mais relevantes da rede neural, testando ordenadamente suas representatividades em relação à qualidade de uma solução (SCHETININ, 2003), conforme ilustrado na figura 2.6. Primeiramente é avaliada a relevância de todas as entradas (x_n) (não ilustrado), obtendo a entrada x_{36} como mais relevante, seguido de x_{23}, x_{10} e x_{60}. Inicialmente é criado o neurônio z_1 conectado a x_{36} e x_{23}. Após, z_2 é criado

conectado novamente a x_{36} e x_{23}, além de z_1. Como o candidato z_2 não aumentou a qualidade da solução, o neurônio é descartado, e a próxima entrada (x_{10}) é utilizada na próxima iteração. Novamente z_2 é criado conectado a z_1, x_{36} e x_{10}, melhorando a qualidade da solução e, portanto sendo consolidado, e assim por diante. Quando todas as entradas foram utilizadas, o algoritmo é interrompido.

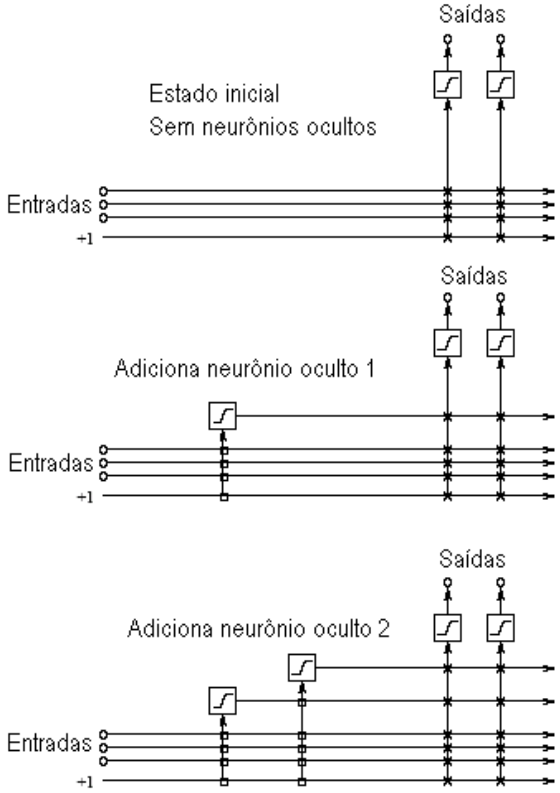


Figura 2.5: Método de aprendizagem em cascata

(Cascade-Correlation Neural Network) (FAHLMAN, 1991, p. 4)

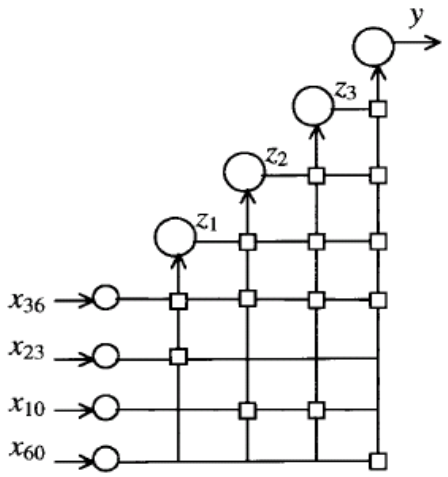


Figura 2.6: Algoritmo de aprendizado para redes neurais evolutivas em cascata

(SCHETININ, 2003)

3 EM BUSCA DE UMA NOVA ABORDAGEM

Nessa seção são analisados os principais detalhes arquiteturais de redes neurais e algoritmos de treinamento até então vistos, entre outros, com o objetivo de chegar a conclusões sobre as principais vantagens de cada método e projetar um novo método híbrido com alta probabilidade de sucesso e alta eficiência computacional baseado em métodos de amostragem estatística.

3.1 Topologias Crescentes

O treinamento de redes neurais a partir de uma estrutura mínima mostrou-se crucial ao longo da história para obtenção rápida e precisa de resultados. Estruturas pequenas possuem uma quantidade reduzida de dimensões, diminuindo o espaço de buscas (ELMAN, 1993).

O método *NEAT* utiliza esse conceito na forma de que se inicia o processo com uma população de topologia mínima, e novos neurônios e conexões são adicionados de forma não determinística, através de mutações. Devido ao baixo desempenho relatado por Kohl (2009), descrita na seção 2.1.2, é levada em conta a abordagem dos métodos de aprendizagem cascateados, criando um novo neurônio e o otimizando ao máximo a cada iteração, de uma forma determinística.

Refletindo sobre a aplicação do princípio de aprendizado cascateado em ambientes de aprendizado por reforço, pode-se supor que a adição de um único neurônio a cada iteração (ou uma conexão a uma única entrada da rede neural), e seu posterior treinamento, não seja suficiente para obter uma lógica mínima representativa de uma solução (LAHNAJÄRVI, 2002). Outra dificuldade é que para identificar a complexidade mínima da rede neural devem-se treinar todas as possibilidades, partindo da mais simples até a mais complexa. Na prática, é desejado um meio-termo entre uma topologia mínima e uma boa taxa de aprendizado, portanto deve-se definir empiricamente um passo de aprendizado que seja relevante ao problema a ser solucionado, e que minimize razoavelmente o número de dimensões a ser otimizadas a cada iteração.

A figura 3.1 ilustra uma proposta de busca crescente em complexidade (da esquerda para a direita) de soluções. São mostradas três soluções com um neurônio a ser avaliadas, tendo como entrada A, B, ou ambas. O próximo passo é utilizar dois neurônios com diferentes permutações de entradas. O algoritmo segue aumentando o número de neurônios até encontrar uma solução representativa (ou mais representativa do que as outras). O número de dimensões exemplificado na figura refere-se aos pesos das conexões das entradas aos neurônios ocultos, aos *bias* dos neurônios ocultos, aos pesos das conexões entre os neurônios ocultos e o neurônio de saída (supondo uma

única saída) e ao *bias* do neurônio de saída (não ilustrados na figura) (detalhes da arquitetura da rede neural são apresentados na seção 3.2).

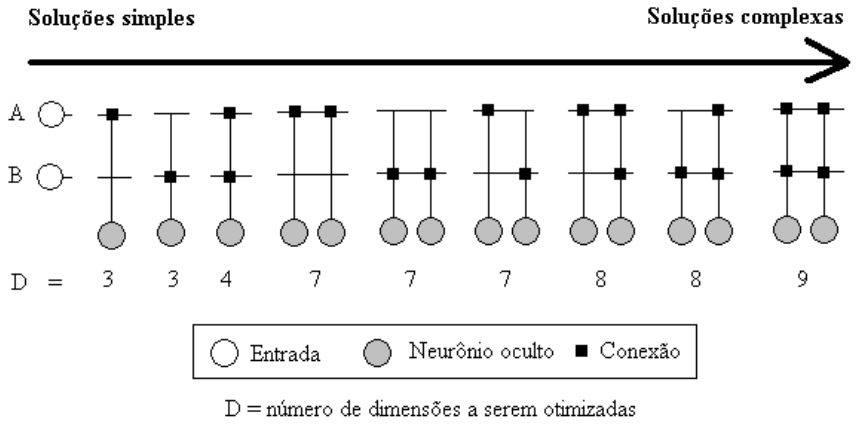


Figura 3.1: Detecção de representatividade para aprendizado por reforço

Outro problema relacionado à detecção de representatividade ocorre no problema de carro-pêndulo. No problema original proposto, o carro inicia exatamente no centro de seu espaço disponível, e dessa forma a solução matemática para o problema de equilibrar o pêndulo não necessariamente depende da posição atual do carro *(x)*, porém assim necessitando uma capacidade de equilibrar o pêndulo muito mais precisa baseada nas outras entradas. Já se for utilizada a posição do carro *(x)*, é mais fácil obter soluções para o problema, porém normalmente o carro acaba se equilibrando em uma distância longe do centro $(x \neq 0)$. Então o problema da representatividade de entradas acabaria encontrando a solução mais complexa, não dependendo da posição do carro *(x)*. Sendo assim, é presumível que testar várias topologias paralelamente é mais aceitável para obter uma solução mais simples, mesmo que com uma topologia contendo mais conexões e neurônios.

3.2 Arquitetura da Rede Neural

Nessa seção é feita uma análise das arquiteturas de redes neurais de *alimentação-avante (feed-forward)* mais comumente utilizadas. Todas elas seguem o princípio do teorema de aproximação universal, que prova que qualquer função pode ser aproximada por um somatório de um peso multiplicado por uma função de transferência continua e monotônica da soma das entradas multiplicadas por respectivos pesos, e adicionalmente um *bias* (CYBENKO, 1989). Devido a detalhes dos algoritmos de treinamento desenvolvidos, foram criadas várias formas de representar uma rede neural.

Na figura 3.2 é representado em *(a)* uma rede neural de uma única camada escondida *(hidden layer)*, podendo conter quantos neurônios escondidos forem necessários para representar a função de saída desejada. Alguns algoritmos de treinamento são automaticamente capazes de criar novos neurônios escondidos conforme a necessidade, outros algoritmos iniciam com um número fixo de neurônios. A rede neural representada em *(b)* é típica dos algoritmos genéticos como o *NEAT*, pois as mutações de criação de novos neurônios e conexões podem acontecer em quaisquer outros neurônios ou conexões, portanto criando redes neurais de múltiplas camadas. A rede neural em *(c)* é do modelo cascateada, onde novos neurônios ocultos são criados conectados a todos os outros neurônios oculto e às entradas da rede neural.

Uma vantagem dessas arquiteturas apresentadas na figura 3.2 é que um mesmo neurônio oculto pode realizar um cálculo relevante para várias saídas, visto que, em um problema de controle, as saídas podem estar relacionadas à identificação de um determinado estado das entradas.

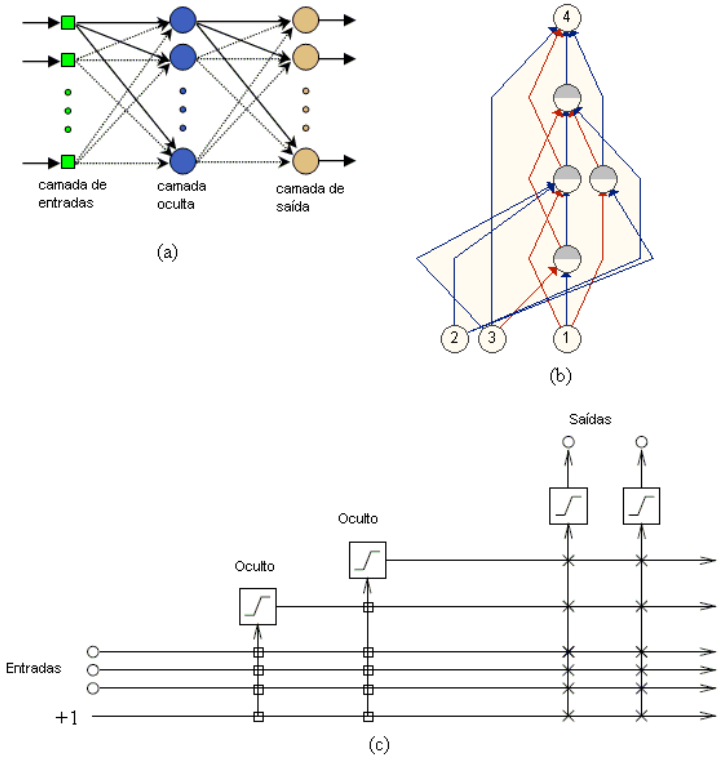
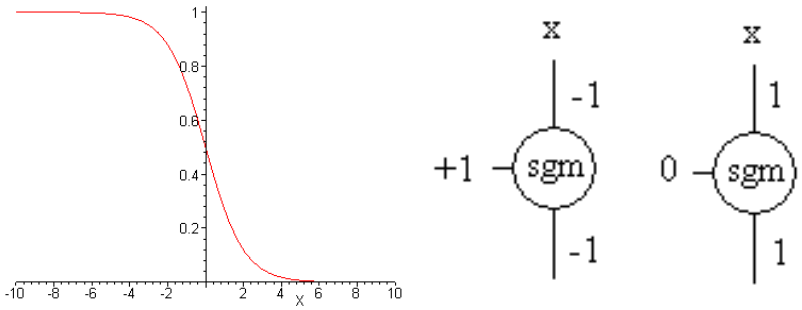


Figura 3.2: Arquiteturas de redes neurais

3.2.1 O Problema da Dupla Representação

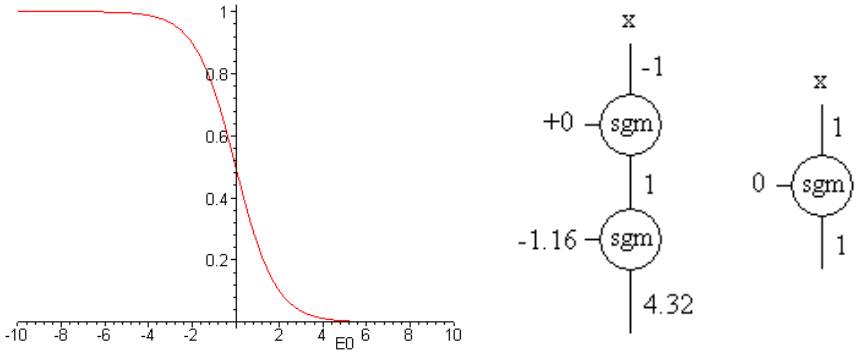
No contexto desse trabalho, onde se deseja realizar amostragem estatística dos pesos e *bias* da rede neural, através da *clusterização* de soluções similares (será abordado na seção 3.4), será vantajoso se não houver dupla representação de redes neurais. Isso significa que não deve haver topologias de neurônios diferentes, tampouco duas configurações de pesos e *bias* de um neurônio, que possam ter como saída a mesma função. As seguintes figuras ilustram algumas situações em que o problema acontece, utilizando a função de transferência sigmoide *(sgm)*.



$$1 \cdot sgm(x+0)+0 = -1 \cdot sgm(-x+0)+1$$

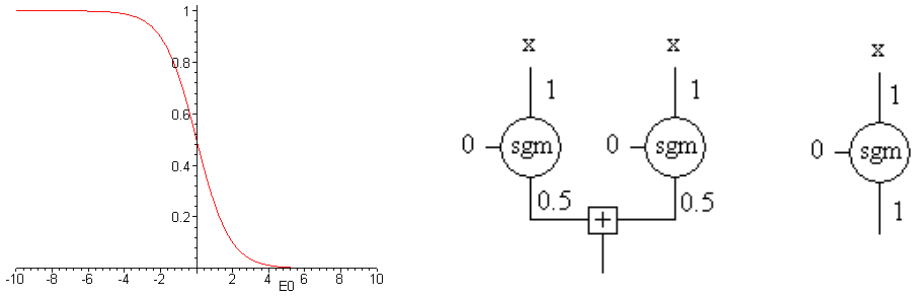
Figura 3.3: Dupla representatividade por pesos e *bias* diferentes

O problema apresentado na figura 3.3 pode ser resolvido utilizando somente pesos positivos nos neurônios, embora isso impeça a utilização de um mesmo neurônio oculto em duas saídas da rede neural, caso uma necessite o peso positivo, e outra necessite o peso negativo.



$$1 \cdot sgm(x+0)+0 \approx -4.32 \cdot sgm(1 \cdot sgm(-x+0))-1.16$$

Figura 3.4: Dupla representatividade estrutural



$$1 \cdot sgm(x+0)+0 = 0.5 \cdot sgm(x+0)+0.5 \cdot sgm(x+0)+0$$

Figura 3.5: Dupla representatividade por soma de componentes

O problema da figura 3.4 e 3.5 seria automaticamente solucionado pela busca em topologias de menor complexidade. No entanto, no método *NEAT*, por as mutações estruturais serem aleatórias, é possível que haja duas topologias equivalentes. O problema da figura 3.5 poderia ocasionar que a inserção de um novo neurônio oculto equivalente (em pesos e *bias)* a um já existente cause uma divergência ao invés da convergência nos valores, pois haveria infinitas possibilidades de pesos que representam a mesma função, dessa forma, seria prudente reduzir os neurônios equivalentes a apenas um, através da soma de seus pesos. Outro problema de infinitas soluções, porém relacionado ao escore, será visto na seção 5.2.1.

Outro problema que surge são neurônios com peso zero na conexão com neurônios de saída, pois não alteram a função realizada. Neurônios que possuem todos os pesos zero em suas conexões com as entradas podem ocasionar um *bias* adicional no neurônio de saída, portanto devem ser eliminados.

3.2.2 A Recombinação de Redes Neurais

A adição de novos neurônios, bem como a recombinação genética utilizados no método *NEAT*, aplicados em redes neurais multicamada, conforme a figura 3.2 *(b)*, causam uma grande alteração na função calculada pela rede neural. A figura 3.6 ilustra a adição de um novo neurônio, da mesma forma que o problema de dupla representação apresentado na figura 3.4. O fator *4,9* utilizado pelo *NEAT* consegue minimizar os efeitos da adição de um novo neurônio, devido ao comportamento linear da função sigmoide utilizando esse fator. Porém é percebida a alteração drástica na função realizada pelos neurônios.

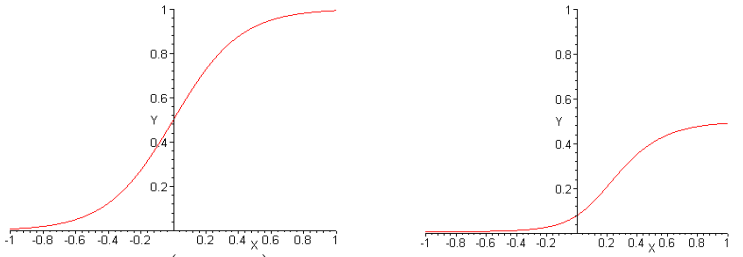


Figura 3.6: Alterações nas funções durante recombinação no método *NEAT*

Essas alterações justificam que o método *NEAT* após realizar uma mutação estrutural também realize mutações nos pesos e *bias* da rede neural, levando em conta que a função original da figura 3.6 poderia ser corrigida com pequenas alterações nos pesos e *bias*, experimentalmente pela seguinte função:

$$1.2 \cdot sgm(4.9 \cdot sgm(4.9 \cdot (0.8 \cdot X)) - 2.5) - 0.1$$

Analisando a recombinação de redes neurais de camada simples, utilizando uma função de transferência linear no neurônio de saída, ilustrada na figura 3.7, obtemos resultados ligeiramente melhores, necessitando apenas eventuais ajustes nos pesos.

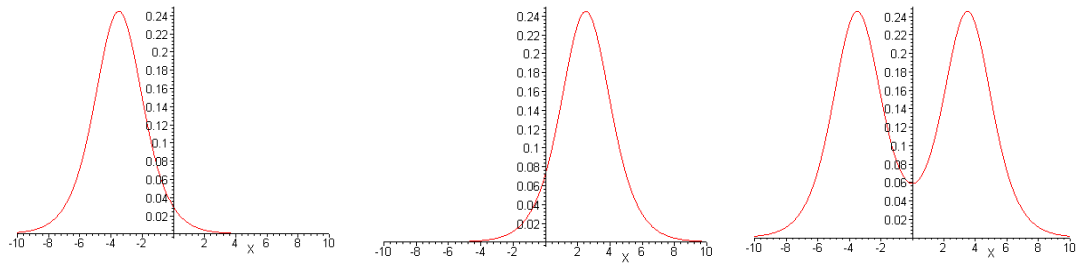


Figura 3.7: Recombinação em redes neurais de camada simples

3.2.3 Uma Abordagem Mais Simples

Devido aos problemas apresentados nas seções 3.2.1 e 3.2.2, nesse trabalho é optada uma implementação mais simples, utilizando neurônios de saída com função de transferência linear, conforme a figura 3.7, além de não permitir conexões entre neurônios ocultos, devido ao problema apresentado na figura 3.6. Assim, evitando os problemas de dupla representatividade, e obtendo um bom desempenho em eventuais recombinações de redes neurais. A figura 3.8 ilustra um exemplo de tal rede neural, contendo duas entradas, e três neurônios ocultos contendo conexões hipotéticas.

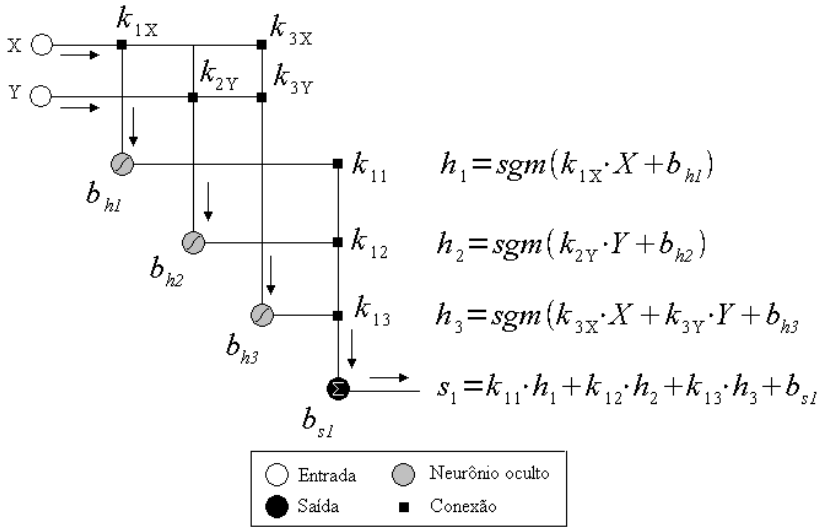


Figura 3.8: Arquitetura de rede neural proposta

3.3 Ótimos Locais

O método de recozimento simulado é capaz de transpor ótimos locais através de sua função de aceitação, que conforme a temperatura do processo é baixada, a probabilidade de aceitação de um escore inferior ao estado atual é diminuída. Desse modo, quando um grande número de candidatos seguidos não for aceito, ou a taxa de candidatos aceitos pelo número de candidatos gerados estiver abaixo de um valor, o algoritmo é encerrado, presumindo que o ótimo global foi alcançado. Os cálculos normalmente são baseados em distribuições de probabilidades predefinidas baseadas na termodinâmica. Se o problema não apresentar tal probabilidade de distribuições, pode ocorrer um baixo desempenho. O algoritmo *Adaptive Simulated Annealing* permite que sejam especificadas outras distribuições.

Já o método *NEAT* elimina espécies que não apresentem melhora de desempenho em um número predefinido de gerações, ou, se nenhuma espécie apresentar melhora de desempenho, somente às duas melhores será permitida a reprodução.

Para evitar o problema de assumir uma distribuição, nesse trabalho será proposta a utilização do armazenamento de um histograma, que é uma técnica estatística não paramétrica, o que significa que ela não assume qualquer predefinição na probabilidade de distribuições. Então, para a geração de candidatos, somente são aceitos candidatos com escore superior a uma grande parcela da população, conforme a figura 3.9 ilustra (para $\rho=0.1$, somente são aceitos candidatos com escore superior a 90% da população).

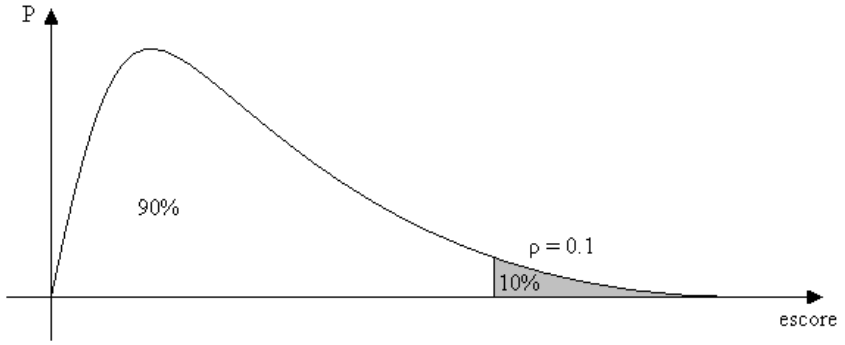


Figura 3.9: Histograma do espaço de busca e taxa de aceitação *(ρ)*

Conforme são gerados candidatos com escore superior, eles são agrupados (conforme a seção 3.4 explica) e seus pesos e *bias* analisados, avaliando a média e desvios-padrão, e assim é esperado um aumento gradual nos escores de um agrupamento. Porém, para realizar a detecção de ótimos locais que devem ser eliminados, será utilizada uma abordagem parecida à do método de recozimento simulado, ou seja, quando a quantidade de candidatos gerados aceitos for menor do que uma parcela de candidatos gerados, é assumido que o agrupamento atingiu um ótimo local. Por exemplo, caso seja utilizado $\rho=0.1$, é esperado que 10% dos candidatos gerados atinjam o escore superior a 90% dos candidatos até então amostrados. Pode-se utilizar uma margem de segurança para evitar a eliminação incorreta do agrupamento, levando em conta que o método é estatístico.

3.4 Nichos de Soluções

O método *NEAT* utiliza a especiação de soluções para a identificação de ótimos locais, bem como para permitir a coevolução de nichos de soluções, levando em conta que diferentes estratégias podem solucionar um problema de controle.

O método de recozimento simulado não possui tal característica, já que o algoritmo objetiva alcançar unicamente o ótimo global, ou uma única solução aceitável.

Nesse trabalho é proposta a utilização de análise de *clusters (cluster analysis)* para agrupamento de soluções similares, equivalentemente a especiação dos métodos genéticos. Há vários algoritmos de *clusterização*, com diferentes graus de determinismo, confiança e custo computacional. Por realizarmos o processo de *clusterização* com poucas amostras, é escolhido um método de fácil implementação, determinístico, de custo cúbico em relação ao número de amostras, o método aglomerativo do vizinho-mais-próximo (*Agglomerative Nearest-neighbour)* (STREHL, 2002).

A explicação do método segue: uma vez que o algoritmo possua uma quantidade predefinida de candidatos com escore superior, é medida a distância entre todos eles (chamados *pontos*), e é iniciada a aglomeração a partir da menor distância. Os pontos são unidos, formando um *cluster*, e a média e desvios-padrão dos pesos e *bias* são calculados. Uma vez que um novo ponto é candidato à aglomeração no mesmo *cluster,* é verificada a distância, segundo alguma métrica (ver seção 3.4.1), para permissão ou recusa da aglomeração. Caso um ponto candidato à aglomeração já esteja inserido em um *cluster*, é avaliada a possibilidade de união dos dois *clusters*. Caso aconteça a recusa da aglomeração, ambos os ponto (ou *cluster)* e *cluster* são considerados terminados, não sendo permitidas mais aglomerações. O algoritmo prossegue até que todas as distâncias calculadas inicialmente sejam testadas, ou todos os *clusters* e pontos estejam finalizados.

É observado que o algoritmo encontra automaticamente o número de *clusters*, e sempre obtém o mesmo resultado caso os pontos sejam avaliados em ordem diferente.

O exemplo da figura 3.10 ilustra um espaço de busca hipotética com duas dimensões *(X* e *Y)* (representando pesos e *bias).* Ocorre a identificação de dois *clusters*, e dois pontos mantêm-se não *clusterizados*. As linhas interligando os pontos representam as distâncias calculadas inicialmente, porém somente as que foram relevantes para o processo, até que todos os pontos e *clusters* foram considerados finalizados.

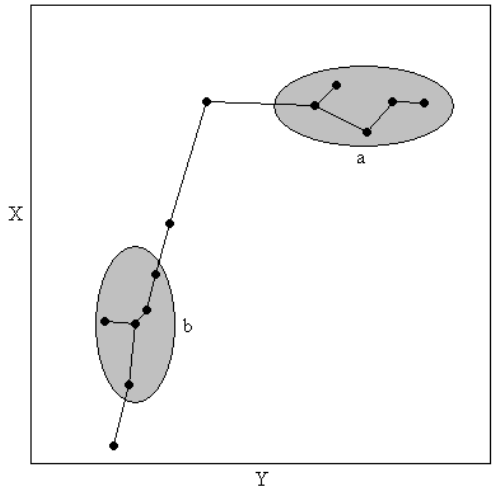


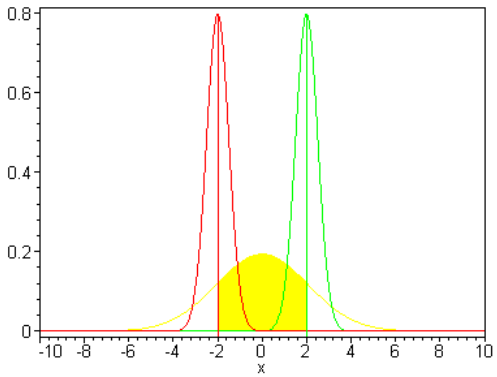
Figura 3.10: Exemplo de *clusterização* proposta

3.4.1 Métrica de Distância Entre Redes Neurais

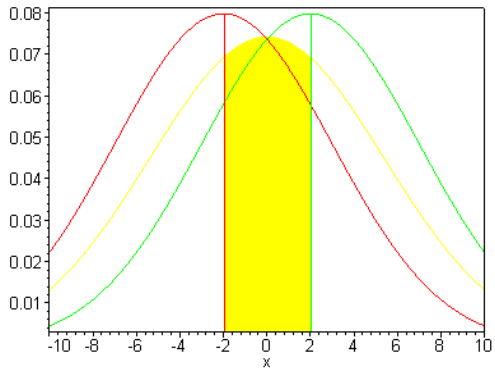
Existem diversas métricas para obtenção da medida de distância em espaços multidimensionais (AGGARWAL), e mais ainda para distâncias entre distribuições normais. Algumas métricas mais avançadas são brevemente vistas na seção 5.3.2.

Para simplificação, e supostamente por eficácia, nesse trabalho, a distância D será medida pela média da soma das distâncias (medida em desvios-padrão, portanto a média deverá ser o valor quadrático médio) entre as distribuições normais das dimensões (pesos e *bias*), expressa pela seguinte fórmula:

$$D = \sqrt{\frac{1}{N} \cdot \sum_{x=1}^{N} \left(\frac{\bar{u}_{1x} - \bar{u}_{2x}}{\sigma(N_{1x} \cup N_{2x})} \right)^2}$$



$D(N(-2,0.5), N(2,0.5)) = 1.94\,\sigma$



$D(N(-2,5), N(2,5)) = 0.74\,\sigma$

Sendo a distância: $D(N_{1,} N_2) = \dfrac{\bar{u}_1 - \bar{u}_2}{\sigma(N_1 \cup N_2)}$

Figura 3.11: Distância entre duas distribuições normais

A figura 3.11 ilustra um exemplo de distância entre duas distribuições normais com a mesma média, porém desvios-padrão diferentes (gráfico da esquerda e da direita). O resultado D é a medida em desvios-padrão (área preenchida) da soma das distribuições.

3.4.2 Distância Entre Redes Neurais de Topologias Diferentes

A arquitetura de rede neural apresentada na seção 3.2.3 permite que seja calculada uma distância entre duas redes neurais de diferentes topologias. Segue a explicação do algoritmo: são calculadas as distâncias entre todos pares possíveis das duas redes neurais, e então é criado o mapa de neurônios mais parecidos entre as duas redes neurais, a partir das menores distâncias.

A figura 3.12 ilustra o caso onde durante a comparação de dois neurônios eles contenham diferentes conexões às entradas da rede neural (*d* falta na segunda rede neural), pode-se assumir que a distribuição normal de pesos da conexão faltante seja zero. Caso as redes neurais contenham diferente número de neurônios, assume-se que os neurônios faltantes possuem peso zero (*b,* conexão com o neurônio de saída), porém assim não são calculadas as distâncias entre os pesos e *bias* das conexões com os neurônios de entrada. Então é realizado o cálculo do valor médio quadrático dos pesos *(a), (b), (c), (d),* e dos *bias (e)* e *(f)*.

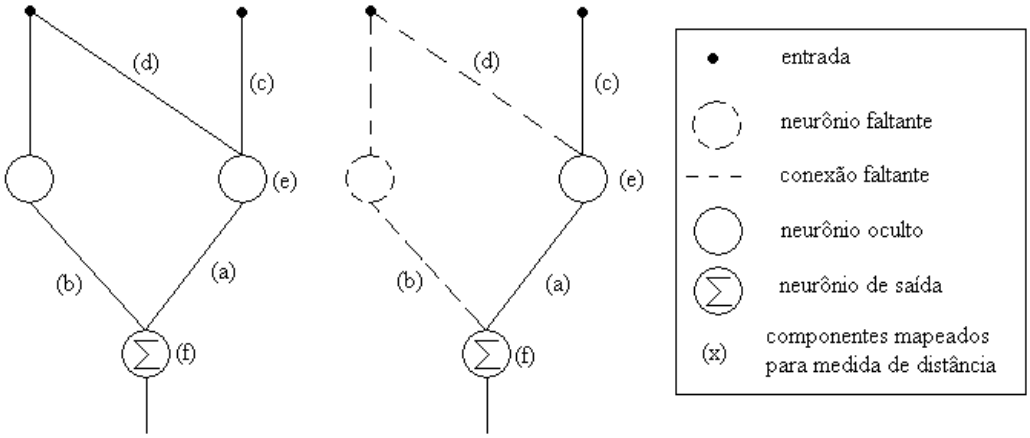


Figura 3.12: Mapeando redes neurais de diferentes topologias

3.5 Gerador de Candidatos

O método *NEAT* não utiliza qualquer heurística sobre os pesos e *bias* dos indivíduos presentes em uma espécie para a geração de novos candidatos (NEUROEVOLUTION, 2010). Já o método *Adaptive Simulated Annealing* possui recursos para amostrar os máximos e mínimos de pesos e *bias* relevantes ao espaço de busca, bem como sua média, permitindo ao usuário definir suas próprias funções geradoras de candidatos, porém, a técnica é mais utilizada como um auxílio para um ajuste inicial (ADAPTIVE, 2010).

Na proposta desse trabalho, há dois momentos em que há a necessidade de obter novos candidatos de alto escore. Num primeiro momento, são obtidos novos candidatos a partir de todo o espaço de busca. Quando são formados *clusters,* são obtidos novos candidatos a partir das médias e desvios-padrão calculados em cada *cluster.* Caso o candidato apresente escore acima da parcela do espaço de busca do *cluster*, ele é incluído nas estatísticas do *cluster* para a geração de novos candidatos. É utilizado o cálculo de médias e desvios-padrão ponderados, ou seja, amostras com maior escore influenciam mais nos resultados.

No momento inicial do treino, quando ainda é desconhecido (ou erroneamente estimado) o valor de alto escore do espaço de busca, devem-se finalizar *clusters* que,

posteriormente à sua criação, venham a ser identificados como de baixo escore. Porém, do mesmo modo, os *clusters* que eram considerados de baixo escore podem vir a ser reabilitados.

Cada *cluster* também mantém estatísticas sobre todos os candidatos gerados, a fim de detectar ótimos locais, conforme explicado na seção 3.3.

A figura 3.13 ilustra o comportamento esperado para a busca de candidatos. *X* e *Y* representam duas dimensões hipotéticas (pesos e *bias)* de uma rede neural. Inicialmente as duas redes neurais representadas pelos pontos na elipse cinza-clara, obtidas pela amostragem de todo o espaço de busca, formam um *cluster.* Em seguida, são obtidos os dois pontos da elipse cinza-média, representando um alto escore em relação à elipse cinza-clara. A partir dos pontos da elipse cinza-média, finalmente é obtido o ótimo local representado pela elipse cinza-escura.

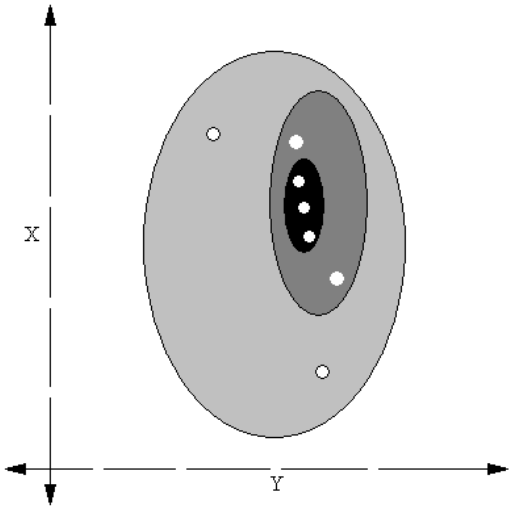


Figura 3.13: Exemplo da geração de candidatos baseada em estatística

3.6 Algoritmo de Aprendizado

O algoritmo proposto é mostrado na figura 3.14. São utilizadas técnicas de sumário estatístico, que não armazenam toda a informação amostrada, exigindo tempo e espaço constante para cálculo de média e desvio-padrão. Os histogramas armazenados podem ocupar um tamanho máximo pré-determinado, porém foi utilizada uma técnica que permite armazenar os valores de amostras somente nas faixas de valores realmente utilizadas. Essas técnicas permitem um grande desempenho computacional, porém não permitem obter os valores originais amostrados, impedindo, por exemplo, que os elementos de um *cluster* sejam *reclusterizados*. As desvantagens dessas técnicas serão discutidas na seção 5.2.3.

```
// Parâmetros de treinamento
ρ  // Parcela da população considerada de alto escore // Conforme seção 3.3
clustering_σ   // distância em desvios-padrão permitidos para clusterização // Seção 3.4
passo_máximo // a quantidade de novos neurônios, ou novas entradas, adicionados a cada
                    passo de treinamento, conforme seção 3.1
// Classes
classe TOPOLOGIA
{
   ESTATÍSTICAS_CANDIDATOS    // de todos os candidatos gerados pela TOPOLOGIA
   CLUSTERS    // Lista de CLUSTER
}
classe CLUSTER
{
   ESTATÍSTICAS_CANDIDATOS    // de todos os candidatos gerados pelo CLUSTER
   ESTATÍSTICAS_MEMBROS       // somente dos candidatos de alto escore
}

// Variáveis globais
TOPOLOGIAS            // Lista de topologias
AMOSTRAS              // Lista de candidatos de alto escore a ser clusterizados

// Algoritmo
Faça
{
   TOPOLOGIAS ← são criadas topologias, ou são adicionados neurônios e entradas conforme
                                                    passo_máximo // Conforme seção 3.1.
   Para cada TOPOLOGIA em TOPOLOGIAS
   {
      se o número de candidatos gerados pela TOPOLOGIA < número desejado // Seção 5.2.2.
      {
         gerar CANDIDATO
         adicionar escore do CANDIDATO às ESTATÍSTICAS da TOPOLOGIA
         se escore do CANDIDATO > ESCORE_ALTO_TOPOLOGIA // Conforme seção 3.3.
            adicionar CANDIDATO às AMOSTRAS
      }
      para cada CLUSTER em TOPOLOGIA
      {
         caso a taxa (número de membros/número de candidatos) > (ρ*margem_segurança)
         {
            o CLUSTER é finalizado // é um ótimo local, conforme seção 3.3.
            prossegue o laço de CLUSTER
         }
         caso o escore do CLUSTER < alto escore da TOPOLOGIA
            finaliza o CLUSTER
            caso contrário, reabilita o CLUSTER // Ver seção 3.5.
         // Opcionalmente pode-se finalizar clusters de baixo escore em relação a outros
clusters.
         // Opcionalmente pode-se favorecer clusters de alto escore em relação a outros
clusters, gerando vários candidatos.
         gerar CANDIDATO de CLUSTER                // Conforme seção 3.5.
         adicionar escore do CANDIDATO às ESTATÍSTICAS_CANDIDATOS do CLUSTER
         se escore do CANDIDATO > ESCORE_ALTO_MEMBROS      // Conforme seção 3.3.
            adicionar CANDIDATO às AMOSTRAS
            adicionar escore do CANDIDATO às ESTATÍSTICAS_MEMBROS do CLUSTER
      }
      caso número de AMOSTRAS da TOPOLOGIA > MIN_AMOSTRAS
         clusterizar CLUSTERS e AMOSTRAS, com desvio-padrão clustering_σ
      caso número de AMOSTRAS da TOPOLOGIA > FINAL_AMOSTRAS & CLUSTERS está vazia
         o treino da TOPOLOGIA é encerrado // ver seção 5.2.2.
   }
   Opcionalmente:
   {
      Para cada TOPOLOGIA em TOPOLOGIAS
         estatísticas da TOPOLOGIA ← Estatísticas do melhor CLUSTER da TOPOLOGIA
      TOPOLOGIAS de escore máximo inferior a outras topologias são eliminadas
   }
   Clusterizar TOPOLOGIAS, com desvio-padrão clustering_σ // Ver seção 3.4.2
   Caso não houve aumento de escore nas topologias, considera-se o algoritmo encerrado
}
```

Figura 3.14: Algoritmo de treinamento

4 TESTES, RESULTADOS E COMPARAÇÕES

4.1 Duplo Carro-Pêndulo

O problema de duplo carro-pêndulo é similar ao apresentado na seção 1.1, porém contendo dois pêndulos de tamanhos diferentes. Deve-se levar em conta o estado inicial do sistema. No problema proposto por Stanley (2001), o carro é inicializado na posição central do espaço, e o pêndulo maior inicializado a um grau em relação à parte superior. Diferentes inicializações podem obter resultados diferentes. A métrica para medida da solução é o número de passos da simulação que ambos os pêndulos mantêm-se equilibrados na parte superior, devendo alcançar 100000 (1000 segundos simulados) para ser considerada uma solução aceita. O código-fonte de simulação foi obtido de uma implementação do método *NEAT* (ANJI, 2005).

A solução para o problema de duplo carro-pêndulo pode ser obtida pela seguinte equação: $F = k_1 \cdot sgm(k_2 \cdot \dot{X} + k_3 \cdot X + k_4 \cdot \dot{\theta} + k_5 \cdot \theta)$ (GEVA, 1993), sendo (*F)* a força aplicada ao carro, (*X*) a posição do carro, ($\dot{X}$) a velocidade do carro, (θ) a posição angular do pêndulo, ($\dot{\theta}$) a velocidade angular do pêndulo, e (k_n) as constantes a ser otimizadas.

Devido à simplicidade da equação solução do problema, é estimado que um único neurônio, tanto no método *NEAT* quanto no método proposto nesse trabalho, será suficiente para obter uma solução. Levando em conta que o método *NEAT* inicia sua população com todas as entradas conectadas ao neurônio de saída, é concluído que o processo é iniciado com a topologia ideal para a solução do problema. Portanto nesse trabalho será utilizada a mesma topologia inicial, para fins de comparação, e, portanto servindo como teste somente para a eficiência da *clusterização* e da busca pelos pesos e *bias*. Foram feitas diversas execuções do algoritmo proposto, para várias configurações de parâmetros. Os resultados podem ser vistos nas tabelas do Apêndice.

Os melhores resultados obtidos foram uma média de 1824 execuções do algoritmo do carro-pêndulo, com erro máximo de ±182 execuções, com 95% de confiança, com desvio-padrão de 1213 execuções, até a obtenção de uma solução, utilizando $\rho=0.1;$ sem eliminação de *clusters* de baixo desempenho; com desvio-padrão para *clusterização* de 2,8; e com um número mínimo de amostras de alto escore para realizar a *clusterização* de uma amostras.

O método *NEAT* necessita em média 3600 execuções da simulação física, com desvio-padrão de 2704 execuções, nenhuma falha, sendo os dados obtidos de 120 execuções (STANLEY, 2001). Assim, assume-se que a média possa ter um erro de ±644 execuções, com 95% de confiança, assumindo uma distribuição normal.

4.2 Carro-Pêndulo

O problema de carro-pêndulo simples possui apenas um pêndulo, sendo, portanto de solução mais fácil. Nesse trabalho, o problema é utilizado de forma mais complexa, devendo equilibrar o pêndulo iniciando de várias configurações iniciais críticas (GEVA, 1993), ou seja: com o carro nos dois extremos do espaço disponível, movimentando-se para fora da pista, com o pêndulo caindo em direção ao centro da pista, e com um ângulo inicial do pêndulo também direcionado ao centro da pista. Esse é o único modo em que o problema é difícil suficiente para que uma busca aleatória não seja suficiente (GEVA, 1993).

Foram utilizadas 2,2 amostras por dimensão, para a minimização de falhas (não encontrar nenhuma solução) conforme a seção 5.2.2 explica. Além disso, foi necessário utilizar um fator de seis vezes para margem de segurança da eliminação de *clusters*, ou seja, se $\rho=0.1$, elimina-se um *cluster* quando a taxa de amostras aceitas pelo número de amostras geradas for $0.1/6$ (conforme seção 3.3). Os resultados podem ser vistos nas tabelas do Apêndice.

A cada situação resolvida dentre as duas configurações iniciais, é esperado um aumento de aproximadamente cinco pontos no escore. Também, em cada situação, o escore pode dobrar proporcionalmente ao tempo em que o carro fica mais próximo do centro do espaço disponível. Portanto, o escore máximo é vinte, e o escore considerado uma solução aceitável foi 19,25.

4.2.1 Evolução da C*lusterização* no Problema de Carro-Pêndulo

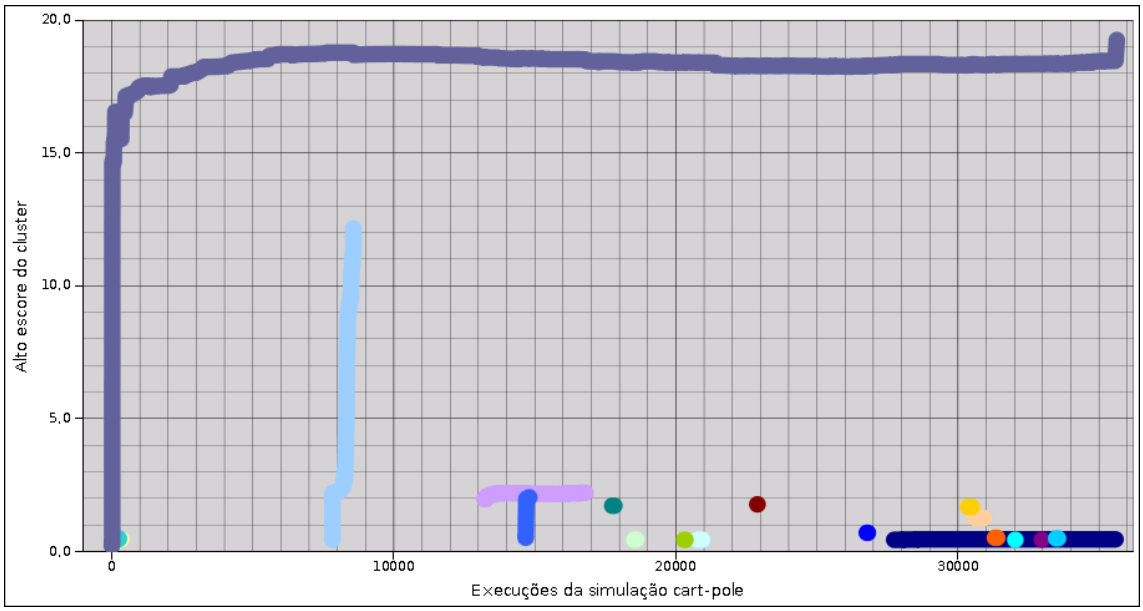


Figura 4.1: Evolução de *clusters* no problema de carro-pêndulo

A figura 4.1 apresenta a evolução dos *clusters* no problema do carro-pêndulo para $\rho=0.1$, sem eliminação de *clusters* de baixo desempenho. No eixo vertical, é representado o escore que é aceito como um alto escore, e não simplesmente o melhor escore até então obtido. Esporadicamente novos *clusters* são gerados, porém são rapidamente absorvidos por seus *clusters* originais.

4.2.2 O Espaço de Busca no Problema do Carro-Pêndulo

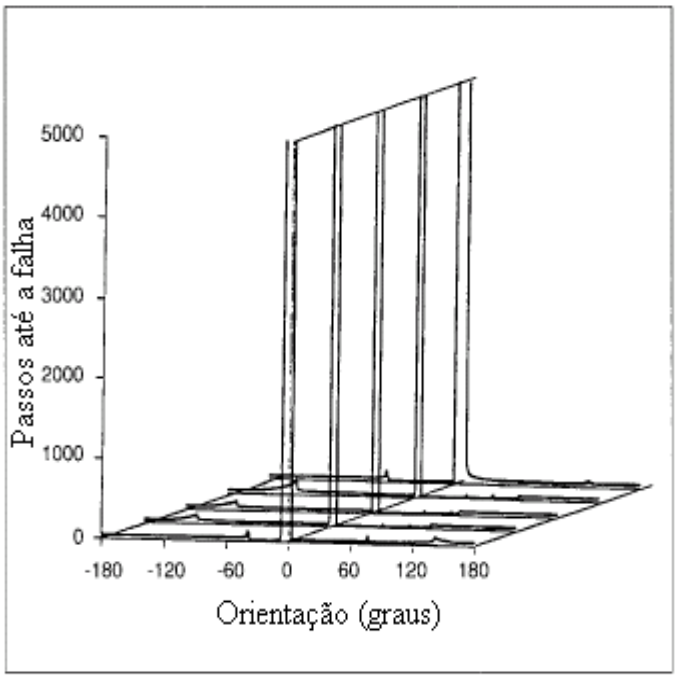


Figura 4.2: Espaço de busca do carro-pêndulo

(GEVA, 1993)

O estudo do problema do carro-pêndulo realizado por Geva (1993) analisou o espaço de busca de forma empírica, obtendo o gráfico da figura 4.2. A figura representa a qualidade da solução (passos até a falha) em função dos pesos e *bias,* porém representando os pesos e *bias* segundo um ângulo. Observa-se que há muitos ótimos locais que devem ser evitados, e há mais de uma solução para o problema (picos no eixo vertical). A definição de uma solução aceitável é que o problema é inicializado em uma posição considerada difícil, e o carro deve centralizar no espaço disponível equilibrando o pêndulo.

5 CONCLUSÃO

Os resultados obtidos nos problemas de carro-pêndulo justificaram a utilização de estatística como uma boa abordagem para problemas de controle. Foram utilizados os princípios de convergência dos métodos de recozimento simulado, combinados com a especiação dos algoritmos genéticos, através da *clusterização* de soluções.

Apesar da analogia com os métodos de recozimento simulado, o algoritmo estatístico possui o princípio de funcionamento diferente. Enquanto o recozimento simulado "percorre" o espaço de busca, tendendo às soluções de alto escore, o método estatístico tende a "conhecer" o espaço de busca, procurando soluções no espaço ainda desconhecido, através da análise de pontos aleatórios (amostras) obtidos.

A utilização dos métodos de aprendizado por reforço transfere o esforço de encontrar uma solução de um problema para a definição da qualidade de uma solução de um problema, visto que o escore é a única fonte de "conhecimento" para a heurística convergir a uma solução ideal. Um exemplo do problema causado por definições ambíguas ou incompletas da função escore é apresentado na seção 5.2.1.

Embora não foi apresentada uma prova de convergência para o algoritmo proposto, é razoável presumir que a utilização de estatística faz o algoritmo convergir de uma forma probabilística, conforme os parâmetros de confiança especificados. Isso deve acontecer desde que as distribuições que são assumidas realmente representem o espaço de busca, ou sejam utilizadas técnicas de estatística não paramétrica, que são vistas na seção 5.3.1. Mesmo os algoritmos de recozimento simulado, que apresentam prova de convergência ao ótimo global, falham (INGBER, 1992). Isso se deve a que os problemas a ser resolvidos não necessariamente possuem as distribuições da termodinâmica assumidas.

Os métodos estatísticos *online* utilizados permitem que os *clusters* armazenem somente as médias e os desvios-padrão dos pontos amostrados, sem necessariamente manter todos os pontos na memória de execução do algoritmo, e atualizando o cálculo a cada ponto inserido em tempo linear. Esses métodos são altamente eficientes, aproximadamente apenas 5% do tempo de computação necessário para resolver os problemas de carro-pêndulo foram utilizados pelo algoritmo de treinamento, e o resto do tempo foi utilizado para a simulação física do problema. Isso abre espaço para a possibilidade de utilizar técnicas mais avançadas, mesmo que mais custosas computacionalmente, visando minimizar o número de simulações do problema. Algumas técnicas proeminentes são vistas na seção 5.3.

5.1 As Contribuições Desse Trabalho

Esse trabalho apresentou diversos problemas decorrentes da utilização de redes neurais multicamada, especialmente o problema de dupla representação, que podem afetar dramaticamente o desempenho dos métodos genéticos. A amenização do

problema da dupla representação e da recombinação de redes neurais através de correções matemáticas; o desenvolvimento de um eficiente gerador de candidatos baseado em amostras das redes neurais obtidas; a fundamentação matemática para a eliminação de ótimos locais; a utilização de topologias crescentes de forma bem controlada; e as métricas de distância entre redes neurais são técnicas que podem imediatamente ser incorporadas nos algoritmos genéticos existentes, devendo ser cuidadosamente testadas em diversas classes de problemas, e validadas ou rejeitadas.

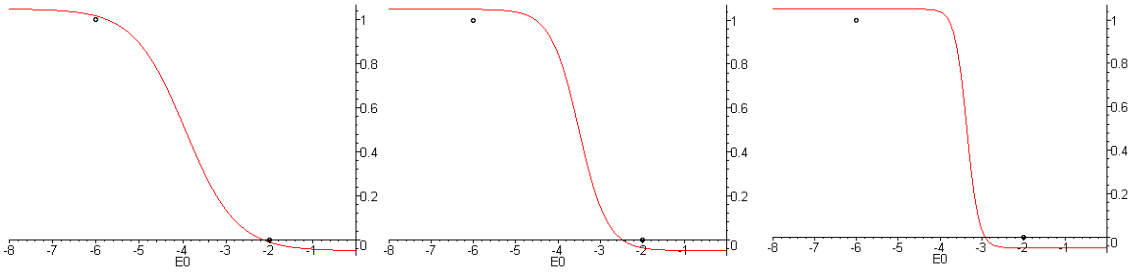
A boa eficiência do método proposto nesse trabalho justifica a utilização de metaotimização estatística, bem como a *clusterização*, devendo servir de inspiração para futuras pesquisas. Embora o método proposto ainda esteja longe de tornar-se um método que resolva qualquer classe de problemas, alguns passos adiante foram dados. A identificação do problema do continuum de soluções (seção 5.2.1), do número de amostras necessários para conhecer o espaço de busca (seção 5.2.2), a identificação das vantagens das topologias cascateadas, o bom desempenho obtido no carro-pêndulo utilizando distribuições normais como heurística, e algumas técnicas que são vistas na seção 5.3 prometem contribuir para a criação de um método altamente adaptativo.

5.2 Os Problemas

5.2.1 Continuum de soluções

A figura 5.1 mostra três soluções que classificam corretamente pontos no problema de N-Pontos, criando assim um "continuum" de soluções podendo chegar a valores infinitos para pesos e *bias* que classificam corretamente os pontos. Isso causou com que o algoritmo proposto divergisse ao invés de convergir para uma solução, criando um número insuportável de *clusters*.

Para a solução desse problema, dois pontos devem ser levados em conta: as distâncias entre a função realizada pela rede neural e o ponto a ser classificado devem ser minimizadas; e os pesos das entradas nos neurônios devem ser artificialmente limitados, ou implementar um mecanismo que detecte a divergência de soluções.



$sgm(1.7 \cdot E_0 + 6.7)$ $sgm(3 \cdot E_0 + 10.5)$ $sgm(7 \cdot E_0 + 23.5)$

Figura 5.1: Continuum de soluções no problema N-Pontos

5.2.2 Amostragem para conhecimento do espaço de busca

O número de amostras necessário até ser obtidos todos os ótimos locais do espaço de busca deve ser levado em conta. O número de amostras cresce exponencialmente com o número de dimensões. Por exemplo, caso seja desejado duas amostras por dimensão a ser otimizada, em uma dimensão necessitamos $2^1 = 2$ amostras, em duas dimensões

$2^2 = 4$ amostras, e assim por diante.

O problema de duplo carro-pêndulo funcionou razoavelmente bem com duas amostras por dimensão, obtendo uma baixa taxa de falhas, indicando que há certa convexidade no espaço de busca. Já o problema do carro-pêndulo, devido a sua maior complexidade (ver seção 4.2) e presença de ótimos locais obteve um número maior de falhas. Assim, a figura 5.1 ilustra um espaço de busca com apenas um ótimo global *(a)* e um espaço de busca com vários ótimos locais *(b)*, bem como a quantidade de pontos necessários para o conhecimento deles.

Pode-se amenizar esse problema aumentando o número de amostras a cada *cluster* encontrado. Assim $amostras = (clusters + 1) \cdot N^{dimensões}$, sendo N o número de amostras por dimensão desejado, devendo ser alto o suficiente para amenizar as chances de uma aleatorização "ruim" ou de "má sorte", que recai em mínimos globais, e não se devem contar amostras que recaiam em *clusters* já conhecidos.

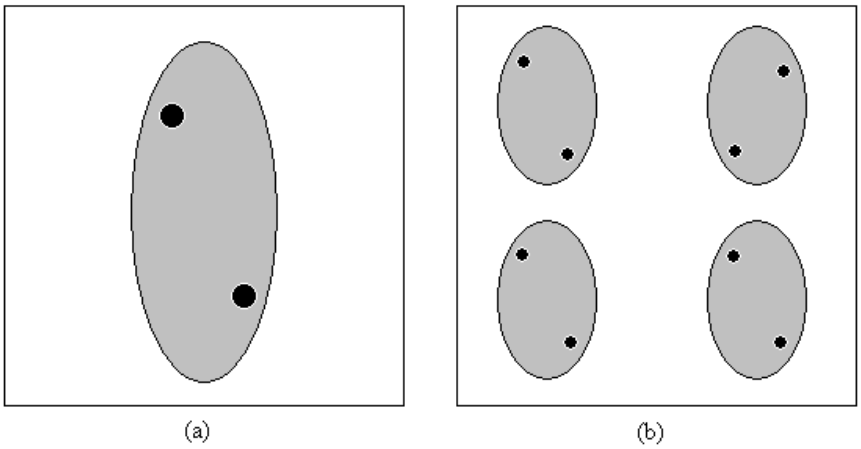


Figura 5.2: Diferente número de ótimos locais do espaço de busca

Além de desse problema, a utilização de ρ (parcela da população aceita como alto escore) pode levar a um treino desnecessário de *clusters* de baixo desempenho.

5.2.3 Perda da informação nos algoritmos de estatística *online*

Os *clusters* utilizados no algoritmo proposto não guardam a informação individual de cada ponto para o cálculo das médias e desvios-padrão dos pesos e *bias*. Dessa forma, caso um *cluster* seja formado contendo dois ou mais ótimos locais, é impossível realizar a *reclusterização* dos pontos em dois *clusters* distintos. A figura na seção 5.2.3 ilustra um caso hipotético dessa situação, onde o *cluster* formado pelos dois pontos brancos não podem convergir aos dois ótimos locais (círculos cinza-escuros).

Outro problema da *clusterização* é que não é levado em conta o escore, apenas a distância entre pesos e *bias*. Isso causa que um ótimo local encontrado na periferia de um *cluster* já existente é incorporado ao *cluster*, causando uma divergência nos valores médios, ao invés de formar um novo *cluster*. A utilização de estatística ponderada também é altamente dependente da fórmula do escore das soluções, assim deve-se levar em conta a utilização de escores quadráticos ou exponenciais, de acordo com o grau de convergência do problema. Esses problemas justificam a necessidade de ajuste diferenciado dos parâmetros ρ e número de amostras e desvios-padrão para *clusterização,* em diferentes problemas, conforme os resultados na seção 4.

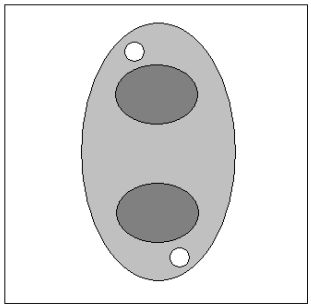


Figura 5.3: Problema da perda de informação, impedindo a *reclusterização*

5.2.4 Busca Inversa a Partir de um Ponto

Durante o processo de amostragem do espaço de estados, é provável que sejam encontradas soluções de alto escore que possam demorar a ser identificadas como parte de um *cluster*, impedindo-as de gerar novos candidatos próximos a si, tendo que esperar pela geração de um ponto próximo ao acaso para a formação de um *cluster*.

Seria viável realizar uma busca inversa, próxima ao ponto, na esperança de identificar um novo *cluster* e um ótimo local. Os desvios-padrão dos pesos e *bias* poderiam ser obtidos do processo de *clusterização,* devendo ser menor do que a menor distância (que é uma média quadrática dos desvios-padrão) até qualquer outro ponto ou *cluster* já amostrado, assim, assegurando, ou tentando evitar, que um novo candidato gerado a partir do ponto não irá sobrepor-se aos outros pontos já amostrados.

5.3 Técnicas Relevantes

5.3.1 Estatística Não Paramétrica

Os métodos estatísticos não paramétricos não assumem que variáveis sigam uma determinada distribuição específica. Esses métodos podem ser ideais para ambientes altamente caóticos, como os problemas de controle.

5.3.1.1 *Método EM (Expectation-Maximization)*

O método *EM* permite criar uma distribuição normal multidimensional correlacionadamente. Essa técnica poderia ser utilizada no gerador de candidatos dos *clusters* no algoritmo proposto. É um algoritmo iterativo que tem custo logarítmico. A figura 5.4 exemplifica as iterações da esquerda para a direita.

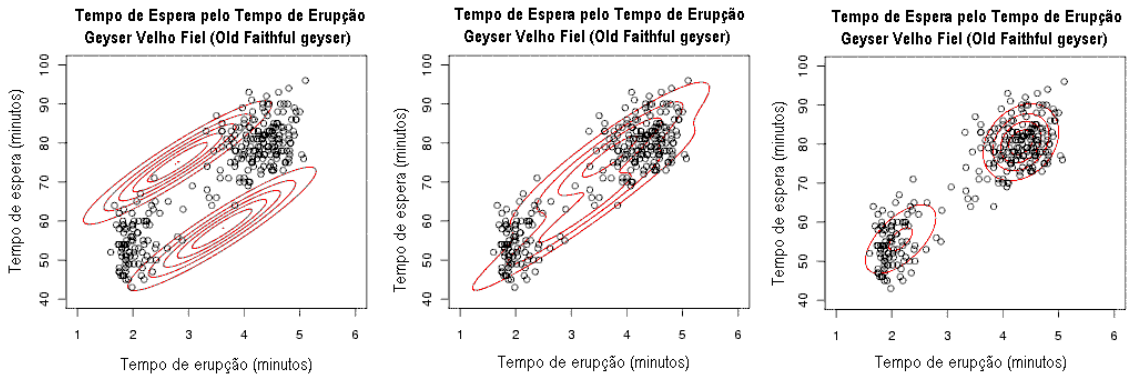


Figura 5.4: Aproximação iterativa do método *EM*

5.3.1.2 *Histograma multidimensional*

Outra abordagem, ao invés de utilizar *clusterização* baseada em distribuições normais, seria utilizar um histograma completo do espaço de busca. Baseando-se na análise do histograma obtido através de amostras, pode-se criar um eficiente gerador de candidatos, tentando explorar espaços próximos a soluções de alto escore, permitindo alcançar e identificar todos os ótimos locais. A figura 5.5 ilustra um eficiente algoritmo adaptativo que mantém aproximações *(b)* de uma amostra real *(a)*.

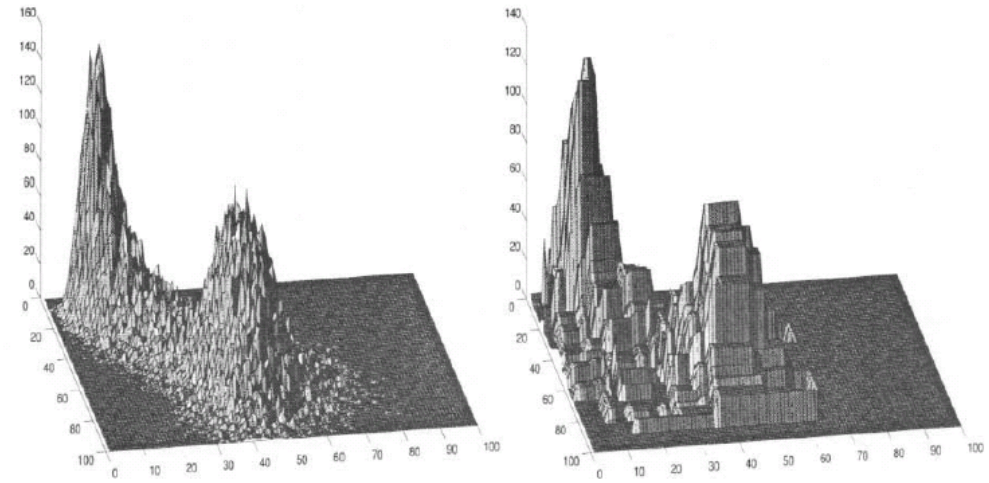


Figura 5.5: Histograma multidimensional dinâmico

(THAPER, 2002)

5.3.1.3 *Método de núcleo para estimativa de densidade (Kernel Density Estimation)*

Tendo um histograma das variáveis, conforme ilustrado na figura 5.5, é necessário identificar os "núcleos" da distribuição, ou seja, é assumido que uma distribuição qualquer é formada pela soma de várias distribuições normais. Sendo obtidas essas componentes normais, elas podem ser utilizadas para uma eficiente geração de candidatos nos pontos ainda desconhecidos. A figura 5.6 ilustra como uma distribuição qualquer pode ser aproximada por um conjunto de distribuições normais, utilizando o método de núcleo.

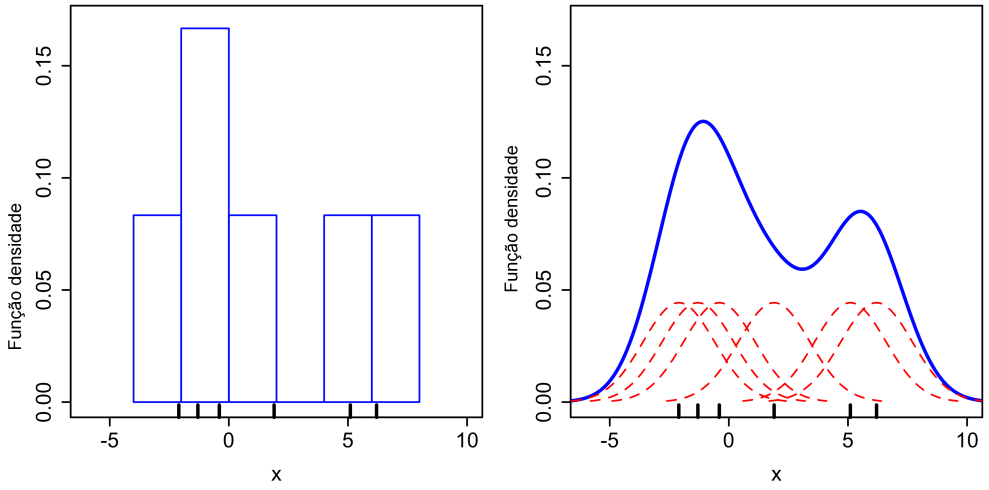


Figura 5.6: Método de núcleo para estimativa de densidade

5.3.2 Métricas de Distância Entre Distribuições Normais Multidimensionais

Um estudo analisou o desempenho de diferentes métricas de distância em espaços multidimensionais para várias classes de problemas (AGGARWAL), obtendo resultados variados em qualidade. A seguir são apresentadas algumas técnicas de estatística paramétrica relevantes ao problema de *clusterização* realizado no método proposto nesse trabalho.

5.3.2.1 *Distância de Mahalanobis*

A métrica de distância de Mahalanobis permite uma análise mais precisa da correlação entre uma distribuição normal em várias dimensões, permitindo calcular a componente "diagonal", conforme a figura 5.7 ilustra.

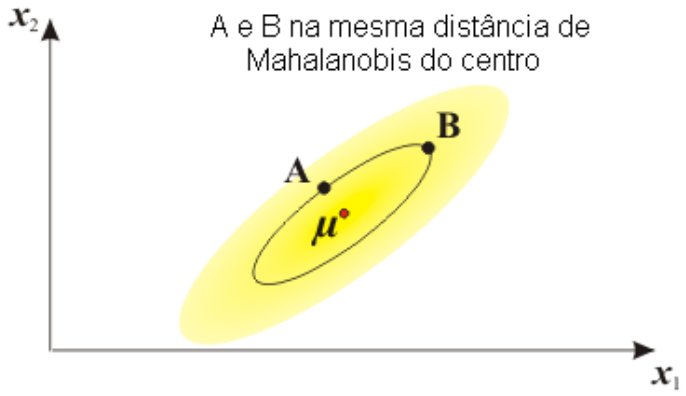


Figura 5.7: Distância de Mahalanobis

O problema ilustrado na figura 5.1 poderia ser facilmente calculado utilizando essa técnica, já que podemos supor que o peso e o *bias* necessários para representação da solução variam de forma linearmente correlacionada.

5.3.2.2 *Distância de Bhattacharyya*

A distância de Bhattacharyya baseia-se no coeficiente de Bhattacharyya, que mede a interseção entre duas distribuições. De uma forma geral, pode-se dizer que para sua utilização em distribuições gaussianas multidimensionais, sua abordagem é parecida à distância de Mahalanobis.

5.3.3 Visualização do espaço de busca

Visualizar o espaço de busca pode ajudar consideravelmente a entender o problema a ser resolvido e o funcionamento do algoritmo de aprendizado.

Os seguintes *sites* mostram o processo de busca dos algoritmos genéticos:

http://www.obitko.com/tutorials/genetic-algorithms/example-3d-function.php

A figura 5.8 ilustra um espaço de buscas hipotético, com a melhor solução (minimizar a função para baixo) e buscas próximas.

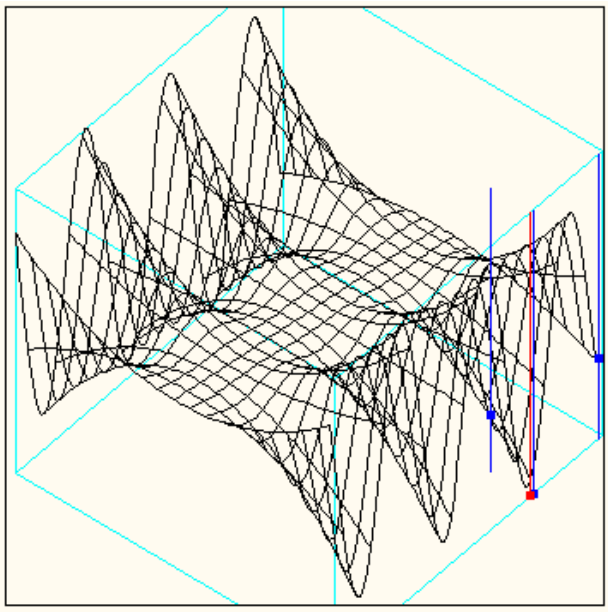


Figura 5.8: Espaço de busca em métodos genéticos

O seguinte *site* mostra o processo de busca do algoritmo de recozimento simulado:

http://mars.wiwi.hu-berlin.de/mediawiki/teachwiki/index.php/Univariate_optimization#Simulated_annealing

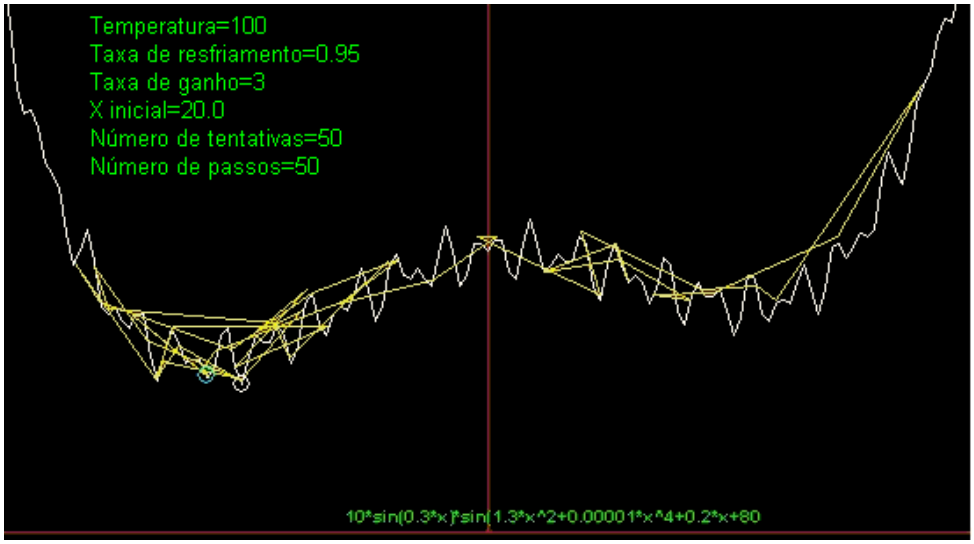


Figura 5.9: Espaço de busca no método de recozimento simulado

A figura 5.9 ilustra um espaço de busca hipotético unidimensional, e a capacidade do método de recozimento simulado em alcançar o mínimo global (para baixo representa solução de maior qualidade).

A visualização de um espaço de busca em uma ou duas dimensões é trivial, porém quando se deseja observar um conjunto maior de variáveis (no caso, o escore em função dos pesos e *bias* da rede neural) é possível utilizar técnicas de redução de dimensionalidade. Um exemplo está no seguinte *site*:

http://www.cs.mcgill.ca/~sqrt/dimr/dimreduction.html

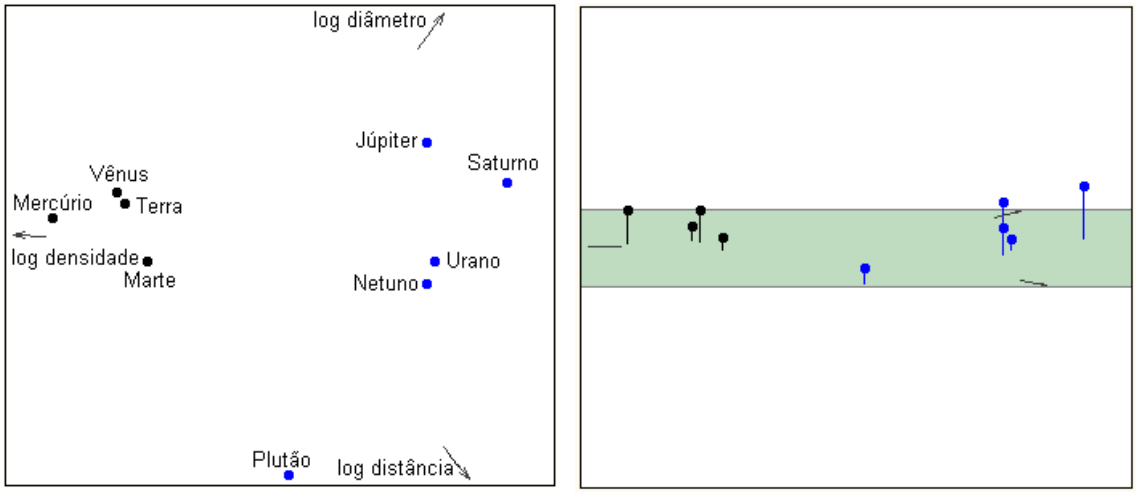


Figura 5.10: O método de redução de dimensionalidade

A figura 5.10 à esquerda ilustra três dimensões sendo projetadas em apenas duas, e mesmo assim, apresentando 97% da variância das componentes (diâmetro, distância ao sol, e densidade). A figura à direita ilustra o erro de representação na terceira dimensão.

5.4 Trabalho Futuro

Foram apresentadas várias dificuldades para a validação do método proposto em ambientes complexos. Uma grande variedade de testes é apresentada na literatura, porém aparentemente cada autor utiliza o teste que mais convém para uma determinada arquitetura, sem importar-se muito nos problemas reais, e tentando promover novos métodos. Deverá ser estudado um conjunto de testes que represente todas as situações possíveis de espaços de busca.

A aplicabilidade de redes neurais em problemas físicos muitas vezes exige que haja conexões recorrentes na rede neural. Isso significa que a rede neural é capaz de analisar o próprio estado das saídas e manter um controle de um "estado atual", além da capacidade de analisar a variação, ou derivada das entradas, e manter um controle temporal. Muitas técnicas realizam o treinamento dessas redes neurais baseando-se na ideia de que deve haver certa estabilidade na rede neural, evitando configurações que sejam caóticas, assim é possível que o simples treinamento proposto nesse trabalho não possa competir com tais métodos. A figura 5.11 ilustra a arquitetura de redes neurais recorrentes mais parecidas à proposta nesse trabalho.

A utilização dos métodos de visualização apresentados na seção 5.3.3 pode ser de ajuda extrema para o desenvolvimento de algoritmos de treinamento, dando uma grande noção intuitiva das técnicas que devem ser aplicadas a cada tipo de problema.

As análises feitas nesse trabalho tem aplicação imediata nos já utilizados métodos genéticos, bem como nos de recozimento simulado, podendo aumentar seus desempenhos. A análise de média e desvios-padrão dos pesos e *bias* de cada espécie pode acelerar e diminuir a necessidade de parâmetros pré-especificados (poder de mutação de pesos) no método *NEAT*, bem como reduzir o espaço de busca nos métodos de recozimento simulado.

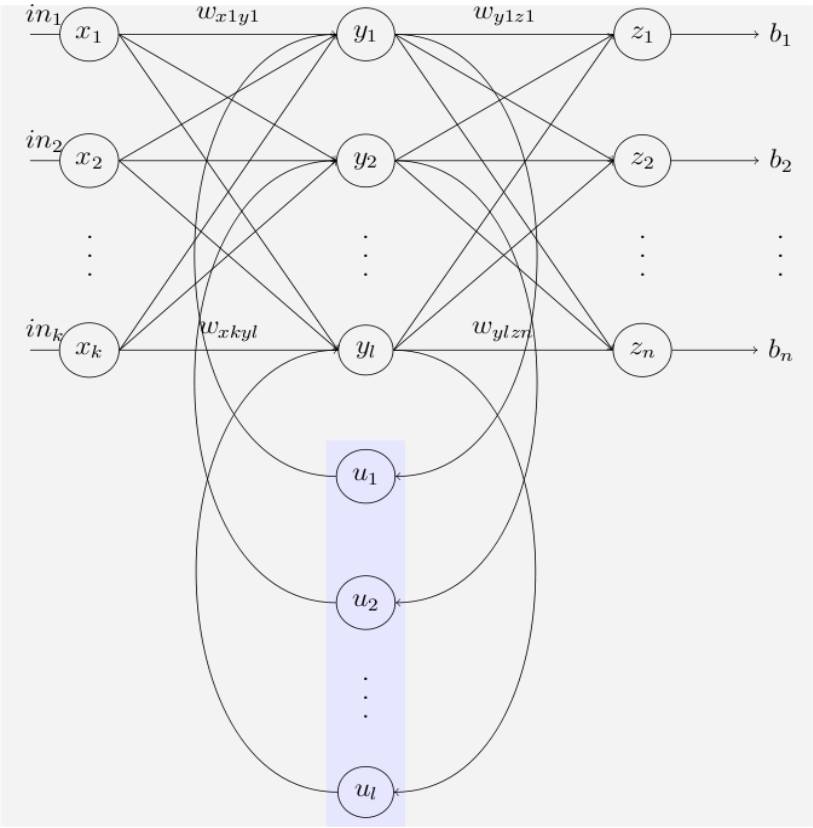


Figura 5.11: Rede Neural Recorrente de Elman

REFERÊNCIAS

ADAPTIVE Simulated Annealing – Readme. Disponível em <http://www.ingber.com/ASA-README.html>. Acesso em: nov. 2010.

AGGARWAL, C. C., Hinneburg, A., Keim, D. A. **On the Surprising Behavior of Distance Metrics in High Dimensional Space.** IBM T. J. Watson Research Center. Yorktown Heights, NY, 10598, USA.

ANJI – Another NEAT Java Implementation. **Sourceforge.** Versão 2.0 (22/08/2005). Disponível em <http://anji.sourceforge.net/>. Acesso em: set. 2010;

BLACK Box. **Wikipedia**. Disponível em <http://en.wikipedia.org/wiki/Black_box>. Acesso em: jun. 2010.

CYBENKO, G. Approximation by Superposition of a Sigmoidal Function. **Math. Control Signal Systems**, n. 2, p. 303-314, 1989.

ELMAN, J. L. Learning and development in neural networks: the importance of starting small. **Cognition, 48.** Departments of Cognitive Science and Linguistics, University of California, San Diego, La Jolla, CA 92093-0526, USA. 1993.

FAHLMAN, S. E; Lebiere, C. **The Cascade-Correlation Learning Architecture.** School of Computer Science, Carnegie Mellon University, Pittsburgh, 1991.

GEVA, S., Sitte, J. A Cartpole Experiment Benchmark for Trainable Controllers. **Control Systems Magazine, IEEE,** v. 13, issue 5, p. 40-51, 1993.

HAYKIN, S. Máquinas Estocásticas e suas Aproximações Baseadas na Mecânica Estatística. In: HAYKIN, S. (Au.) **Redes Neurais: Princípios e prática**. Trad. Paulo Martins Engel. 2.ed. Porto Alegre: Bookman, 2001. p. 591-642.

INGBER, L. **Very Fast Simulated Re-Annealing.** Physics Department, Naval Postgraduate School Monterey, CA 93943 and U.S. Army Concepts Analysis Agency 8120 Woodmont Avenue, Bethesda, MD 20814-2797, 1989.

INGBER, L; Rosen, B. Genetic Algorithms and Very Fast Simulated Reannealing: a Comparison. **Mathematical and Computer Modeling,** n. 16(11), p. 87-100, 1992.

KOHL, N. F; B.A; M.S. **Learning in Fractured Problems with Constructive Neural Network Algorithms**. 2009. 173 f. Thesis (Doctorate in Philosophy) – Faculty of the Graduate School, The University of Texas at Austin.

LAHNAJÄRVI, J. J. T., Lehtokangas, M. I., Saarinen, J. P. P. Evaluation of Constructive Neural Networks With Cascaded Architectures. **Neurocomputing 48**, p. 573–607, 2002.

MAHFOUD, S. W.; Goldberg, D. E. Parallel Recombinative Simulated Annealing: A Genetic Algorithm. **Parallel Computing**, n. 21, p.1-28, 1995.

NEUROEVOLUTION. The Neuroevolution of Augmenting Topologies (NEAT) Users Page. Disponível em <http://www.cs.ucf.edu/~kstanley/neat.html>. Último update: 19/7/2010. Acesso em: nov. 2010.

PADERSEN, M. E. H. **Tuning & Simplifying Heuristical Optimization.** 2010, 204 f. Thesis (Doctorate in Philosophy) – School of Engineering Sciences, University of Southampton.

RAM, D. J; Sreenivas, T. H; Subramaniam, K. G. Parallel Simulated Annealing Algorithms. **Journal of Parallel and Distributed Computing 37**, n. 121, p. 207–212, 1996.

SCHETININ, V. A Learning Algorithm for Evolving Cascade Neural Networks. **Neural Processing Letters** 17, p. 21–31, 2003.

STANLEY, K. O., Miikkulainen, R. **Evolving Neural Networks through Augmenting Topologies.** 2001. Department of Computer Sciences, The University of Texas at Austin, Austin, TX.

STANLEY, K. O., Bryiant B. D., Miikkulainen, R. Evolving Neural Network Agents in the NERO Video Game. 2005. **Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games.**

STHREL, Alexander. **Relationship-based Clustering and Cluster Ensembles for High-dimensional Data Mining.** 2002, 214 f. Dissertation (Doctorate in Philosophy) – Faculty of the Graduate School of The University of Texas at Austin.

SUTTON, R. S.; Barto, A. G.; Anderson, C. W. Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. 1983. **IEEE Transactions on Systems, Man and Cybernetics**, Vol. SMC-13, n. 5, p. 835-846.

THAPER, N. et al. Dynamic Multidimensional Histograms. **ACM SIGMOD '2002 June 4-6**, Madison, Wisconsin, USA. 2002.

ANEXO – ARTIGO DA PROPOSTA DESSE TRABALHO

Algoritmo de Aprendizado para Ambientes Complexos Baseado em Recozimento Simulado Paralelo e Redes Neurais em Cascata Utilizando Metaotimização Estatística

Jorel Settin[1], Paulo M. Engel[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

{jsettin,engel}@inf.ufrgs.br

Resumo. *Este trabalho apresenta um algoritmo de aprendizagem para redes neurais com objetivo de alcançar soluções ótimas com grande probabilidade em problemas onde não se tem o conhecimento prévio da solução ótima a ser atingida e da complexidade da rede neural necessária. Utiliza-se estatística como metaotimização para obtenção automática dos parâmetros dos algoritmos de Recozimento Simulado Paralelo (Parallel Simulated Annealing) e de Redes Neurais em Cascata (Cascaded Neural Network), analisando a variabilidade das soluções encontradas. A eficiência do algoritmo é verificada no problema de aproximação de funções.*

1. Introdução

Este trabalho apresenta uma proposta de algoritmo de aprendizado de redes neurais para situações complexas e de incerteza onde há soluções diferentes que podem atingir a qualidade desejada, onde a complexidade da rede neural necessária para alcançar uma solução é desconhecida, e onde a qualidade de soluções é dificilmente avaliada devido a complexidade do ambiente ou problema simulado [Black Box 2010]. Essas situações podem ter de exemplo a natureza, onde são encontradas soluções diversas como as diferentes maneiras que os animais desenvolveram para caminhar ou correr eficientemente [Basso 2005], sendo uma área inspiradora para a robótica. Pode-se citar a complexidade algorítmica de estratégias de jogos, onde a complexidade computacional exigida para a obtenção de uma estratégia inicial é desconhecida [Neyman 2003] [Stanley 2005], e sistemas caóticos, como aerodinâmica e outros sistemas físicos, que podem variar bruscamente a qualidade de uma solução com pequenas mudanças.

No corrente estado da arte, têm sido propostas várias arquiteturas de redes neurais e de algoritmos de aprendizagem, conseguindo-se bons resultados para aplicações específicas. Porém são demonstradas várias dificuldades em ambientes complexos, e mesmo métodos como algoritmos genéticos tem se mostrado ineficientes por apresentar uma baixa variabilidade na procura de soluções [Kohl 2009], justificando a utilização do método probabilístico de Recozimento Simulado (Simulated Annealing) onde a busca por soluções tem a variabilidade controlada dinamicamente pelo parâmetro “temperatura”.

A arquitetura de Redes Neurais Evolutivas em Cascata (Evolving Cascade Neural Networks - ECNN) tem se mostrado promissora para obter soluções aceitáveis com número mínimo de neurônios e conexões, por primeiramente avaliar as entradas mais relevantes de um sistema [Schetinin 2003], portanto é uma escolha razoável como

base desse trabalho.

Devido à tecnologia atual, a utilização de computação massivamente paralela tem sido amplamente utilizada. Assim, algoritmos paralelos de aprendizado têm sido propostos, justificando a utilização de populações para obter uma grande variabilidade de soluções bem como a paralelização do algoritmo [Ram 1996].

Para diminuir a complexidade computacional dos métodos de aprendizado e a dificuldade em escolher os parâmetros ideais dos algoritmos, como tamanho de populações e o grau de variação dos parâmetros da rede neural (pesos e limiares), são utilizados métodos de metaotimização, sendo o método estatístico uma escolha aceitável para ambos os problemas [Padersen 2010].

A eficiência do algoritmo proposto será analisada através do teste de aproximação de funções por erro quadrático médio, que é amplamente aceito como método para avaliar algoritmos de otimização [Root Mean Square Deviation 2010]. Para simular uma situação onde há várias soluções aceitáveis, é proposto um teste onde a rede neural deve aproximar uma dentre várias funções pré-determinadas.

2. Análise de Arquiteturas de Redes Neurais para Ambientes Complexos

Na seção 2.1 será apresentado o algoritmo genético NEAT, o qual tem sido utilizado para o treinamento de estratégia de jogos de operações robóticas (Neuro Evolving Robotic Operatives – NERO) por apresentar soluções intuitivamente criativas [Stanley 2005]. Em seguida, na seção 2.2, é apresentado um estudo que sugere que o método NEAT não apresenta um grau de variabilidade suficiente para obter a melhor solução com alta probabilidade [Kohl 2009]. Devido a essa contradição da eficiência da arquitetura NEAT, nesse trabalho recorre-se aos métodos não genéticos amplamente utilizados que descobrem sistematicamente a relevância das entradas da rede neural, sendo esperado obter uma rede neural de tamanho mínimo e próxima ao ótimo global, sendo apresentada a arquitetura evolutiva em cascata na seção 2.3 [Schetinin 2003].

Na definição da implementação desse trabalho (seção 3) é então proposta uma adaptação das arquiteturas evolutivas em cascata para ambientes complexos, justificando que o método até então apresentado assume que uma única entrada da rede neural ou um único neurônio sempre apresenta uma solução significativa e mensurável. É realizada uma analogia entre o método genético e o método sistemático apresentado, tentando utilizar as melhores características dos dois métodos para ambientes complexos.

2.1. Neuroevolução de Topologias Aumentativas (NeuroEvolution of Augmenting Topologies – NEAT)

Os métodos genéticos se baseiam nos princípios de recombinação e mutação genética de indivíduos de uma população candidatos à solução do problema. Os genes (genótipo) contém uma representação de uma rede neural (fenótipo). Através das gerações, acontecem recombinações de genes e mutações, sendo esperado que uma solução melhor seja encontrada.

O primeiro motivo pelo qual o método NEAT tem se mostrado promissor em buscas de estratégias complexas para jogos é que ele possui um algoritmo base de recombinação genética, onde os genes que representam neurônios possuem uma numero identificador passado através das gerações, permitindo realizar a recombinação de conexões e neurônios realmente relacionados entre si e assim aumentando as chances de

manter as melhores topologias da rede neural. A figura 1 ilustra o algoritmo de recombinação.

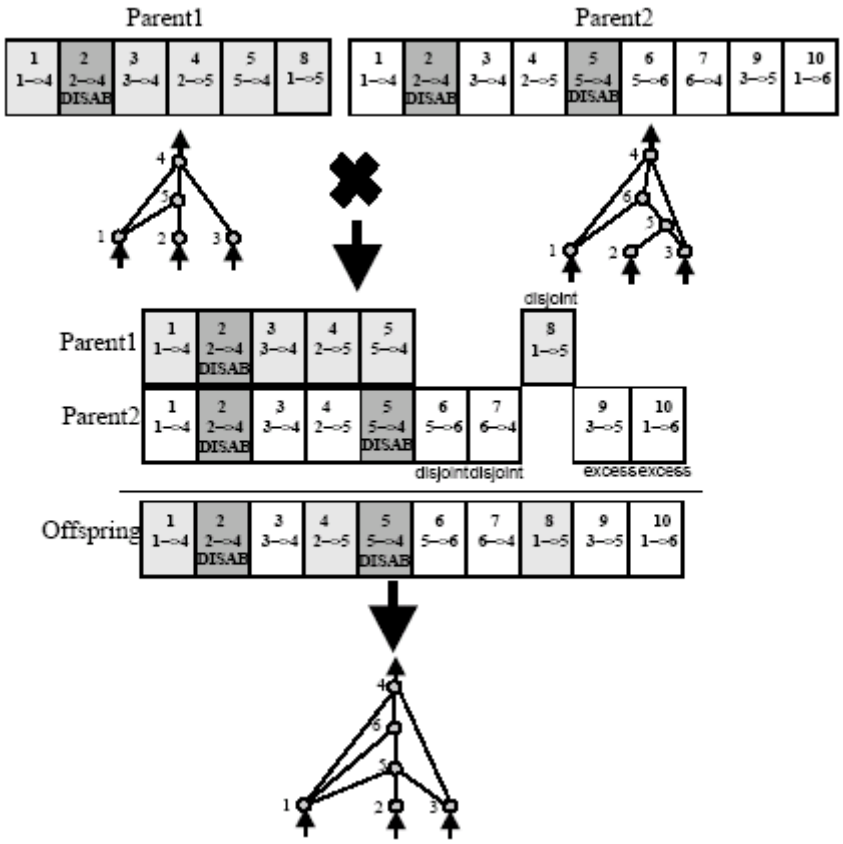


Figura 1. "Casando genomas de diferentes topologias usando *números de inovação.* Embora o *Pai 1* e o *Pai 2* pareçam diferentes, seus *números de inovação* (mostrados na parte superior de cada gene) nos mostra quais genes estão relacionados entre si. Mesmo sem qualquer análise topológica, uma nova estrutura pode ser criada, combinando as *interseções* dos dois pais bem como suas partes diferentes. Genes relacionados são herdados randomicamente, enquanto *genes disjuntos* (aqueles que não casam, mostrados no meio) e *genes em excesso* (aqueles que não casam, mostrados ao final) são herdados do *Pai* mais adaptado. Nesse caso, igual adaptação é assumida, então os *genes disjuntos* e os *genes em excesso* são também herdados randomicamente. Os *genes desabilitados* podem se tornar habilitados novamente em futuras gerações: existe uma chance pré-determinada que um gene herdado esteja desabilitado se ele está desabilitado no pai também." [Stanley 2001]

O segundo motivo pelo qual o método NEAT encontra soluções complexas é porque, partindo de uma estrutura mínima, incrementalmente adiciona novos neurônios e conexões. A figura 2 ilustra esses dois tipos de mutação.

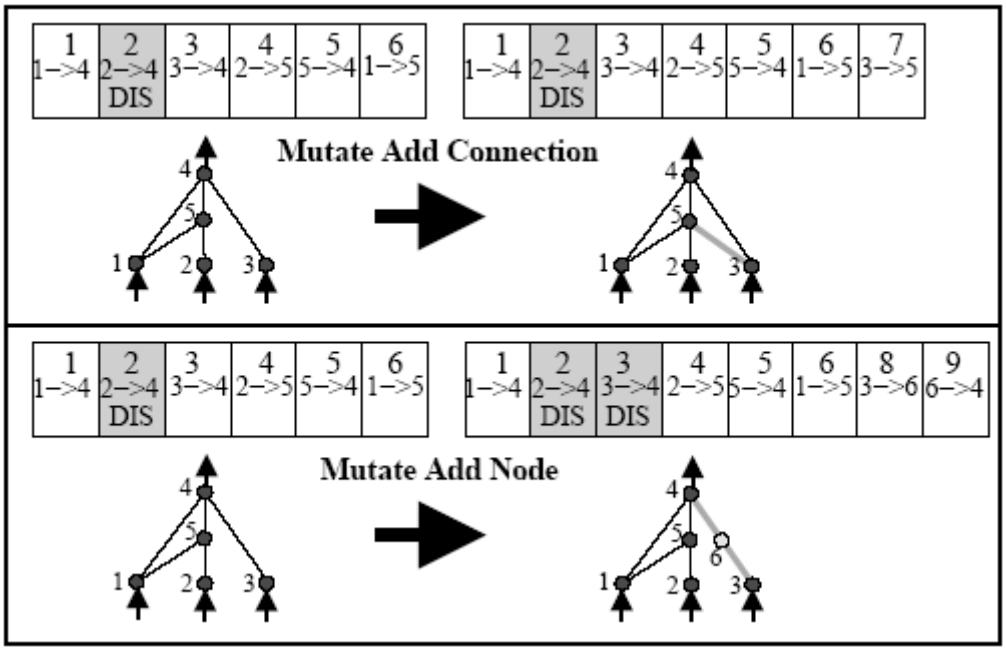


Figura 2. "Os dois tipos de mutação estrutural no NEAT. Ambos os tipos, adicionando uma conexão e adicionando um nodo, são ilustrados com os *genes conectores* de uma rede neural, mostrados acima de seus fenótipos. O número superior em genoma é o *número da inovação* daquele gene. Os *números de inovações* são marcadores históricos que identificam o ancestral histórico original de cada gene. Para os novos genes criados o número é incrementado. Adicionando uma conexão, um simples nono *gene de conexão* é adicionado ao fim do genoma dado o próximo *número da inovação* disponível. Adicionando um nodo, o *gene de conexão* sendo dividido é desabilitado, e dois novos *genes de conexão* são adicionando ao fim do genoma. O novo nodo fica entre as duas novas conexões. Um novo *gene de nodo* é adicionado ao genoma." [Stanley 2001]

Através de sucessivas pequenas mutações na topologia e nos pesos da rede neural, o algoritmo NEAT se mostrou eficiente para topologias relativamente pequenas, sendo suficiente para criar uma estratégia iterativa em jogos. Porém, um estudo apresentado na próxima seção (seção 2.2) demonstra as limitações do algoritmo quando é exigida uma topologia mais complexa.

2.2. Avaliação do NEAT em Problemas Fraturados

Uma avaliação do algoritmo NEAT problemas que exigem um alto grau de variabilidade das soluções demonstrou sua baixa eficiência nesses ambientes [Kohl 2009].

A figura 3 ilustra o problema da classificação de pontos em duas situações, uma com baixa variabilidade, outra com alta.

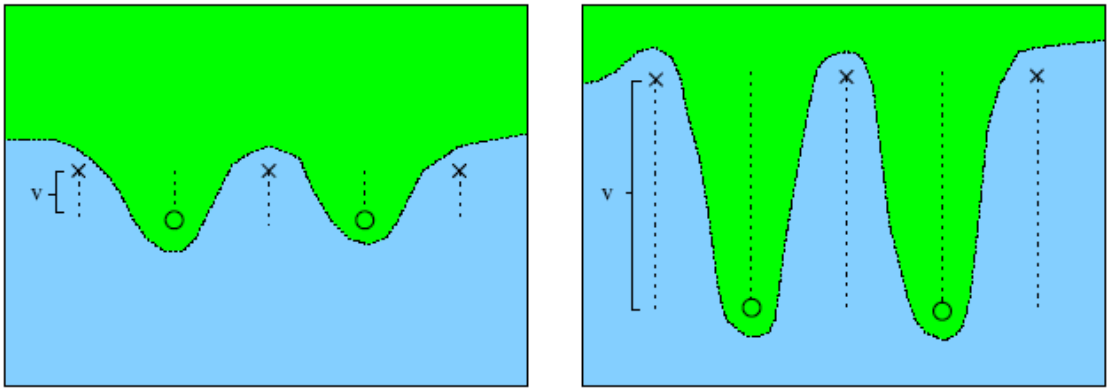


Figura 3. "Exemplo de soluções para o problema de N-Pontos com N=5 e a separação entre as duas classes de pontos é *(a) v=0.1* e *(b) v=0.8*. Conforme *v* aumenta, as duas classes de pontos movem-se mais longe uma da outra, e a variação mínima necessária para descrever a fronteira entre as duas classes aumenta." [Kohl 2009]

A figura 4 ilustra os resultados do desempenho do algoritmo NEAT para os diferentes graus de variabilidade exigidos.

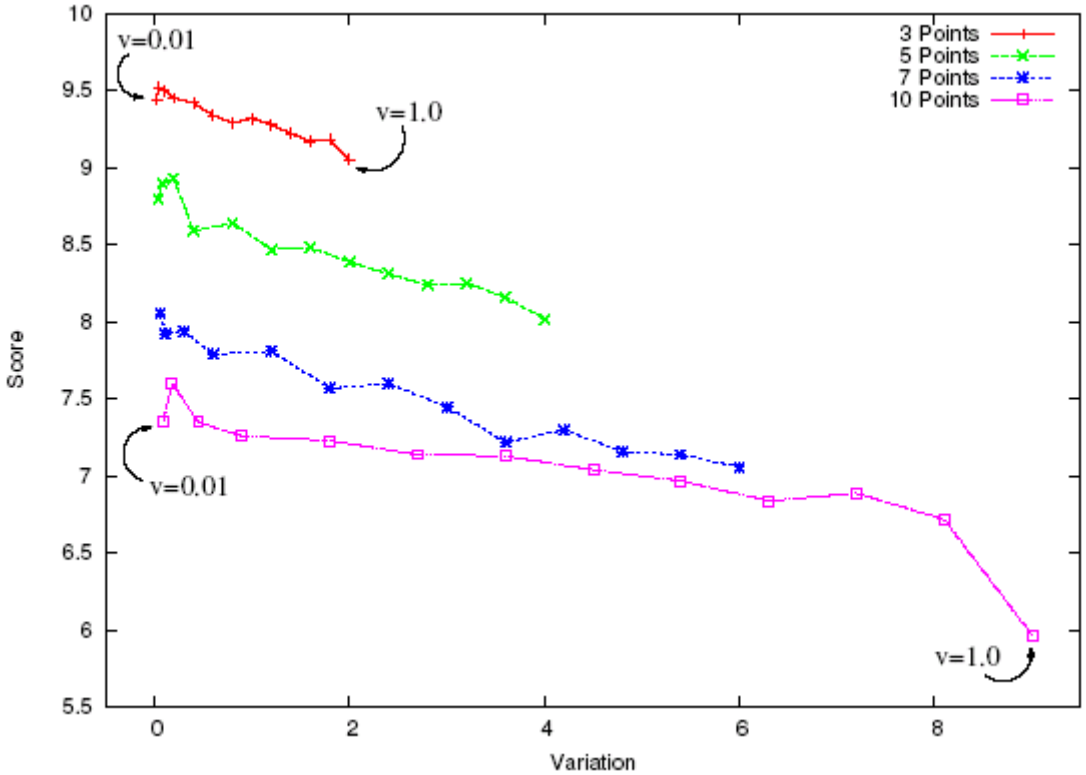


Figura 4. "Desempenho do NEAT no problema de N-Pontos para *N = 3, 5, 7, 10* e *v = 0.01..10.* Conforme *v* aumenta, a variabilidade necessária para resolver o problema também aumenta, e o desempenho do NEAT cai. O mesmo ocorre conforme *N* aumenta. Esse resultado realça a dificuldade do NEAT em produzir soluções com alta variabilidade." [Kohl 2009]

2.3. Redes Neurais Evolutivas em Cascata (Evolving Cascade Neural Network – ECNN)

Ao contrário dos algoritmos genéticos, os métodos de aprendizado de redes neurais em cascata buscam sistematicamente soluções, partindo de um único neurônio e aumentando gradualmente. Tais métodos classificam a relevância de cada entrada da rede neural, inserindo novos neurônios utilizando primeiramente as entradas mais relevantes para evitar que a rede neural seja treinada por ruídos das entradas, dificultando o treinamento com as entradas relevantes, e terminando quando a solução atinge a qualidade desejada [Schetinin, 2003].

O seguinte pseudoalgoritmo é proposto na arquitetura ECNN:

(1) Calcular a adaptabilidade para um neurônio conectado a cada entrada da rede neural.

(2) Ordenar os valores calculados e salvá-los em uma lista, sendo a primeira entrada a mais relevante.

(3) Número da entrada $h \leftarrow 2$.

(4) Treinar um novo neurônio candidato conectado aos neurônios previamente inseridos, à primeira entrada, e à entrada *h*.

(5) Se a qualidade da solução no passo 4 piorou, $h \leftarrow h+1$ e voltar ao passo 4, ou terminar caso *h>número de entradas*.

(6) Caso contrário, adicione o neurônio treinado e volte ao passo 4, mantendo o valor de *h*.

A figura 5 ilustra uma rede neural treinada por esse algoritmo.

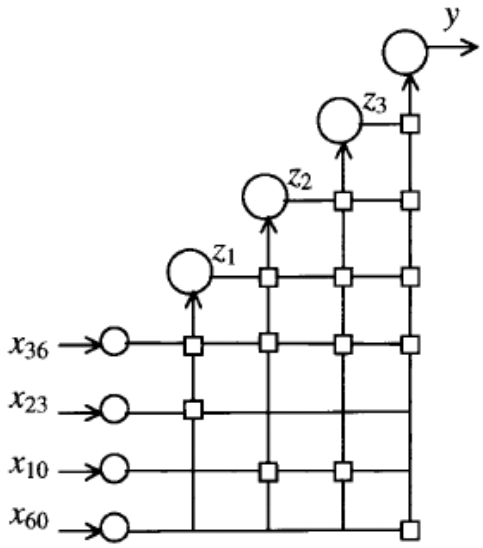


Figura 5. Primeiramente avalia-se a relevância de todas as entradas x_n (não ilustrado), obtendo-se a entrada x36 como mais relevante, seguido de *x23*, *x10* e *x60*. Inicialmente é criado o neurônio *z1* conectado a *x36* e x23. Após, *z2* é criado conectado novamente a *x36* e x23 , além de z1. Como o candidato *z2* não aumentou a qualidade da solução, o neurônio é descartado, e a próxima entrada (*x10*) é utilizada na próxima iteração. Novamente *z2* é criado conectado a *z1*, *x36* e *x10,* melhorando a qualidade da solução e portanto sendo consolidado, e assim por diante. Quando todas as entradas foram utilizadas, o algoritmo é interrompido.

O método apresentado mostra-se eficiente pois treina a rede neural inicialmente com as entradas mais relevantes, evitando ruídos.

Analisando o método, espera-se que ele falhe em ambientes complexos pelos seguintes motivos: em ambientes complexos, a relevância de uma única entrada de um sistema pode ser insignificante em todos os casos, pois não se alcança um nível mínimo necessário de complexidade para obter algum resultado mensurável, e assim o algoritmo apresentado falharia em ordenar a relevância de cada entrada; em ambientes complexos, a inserção de apenas um neurônio candidato em cada iteração pode ser insignificante na qualidade da solução, pelo mesmo motivo do item anterior.

Por esses motivos, é proposta uma adaptação desse algoritmo para ambientes complexos na seção 3.

3. Proposta de Algoritmo de Aprendizado para Ambientes Complexos

Na seção anterior pode-se concluir que o método NEAT é capaz de evoluir a topologia de redes neurais, criando novos neurônios e conexões, e assim aumentando a complexidade da solução. Porém, o método NEAT falha na capacidade de gerar alta variabilidade de soluções, existindo um parâmetro pré-determinado indicando a mutação máxima dos pesos sinápticos e do crescimento da topologia. Em outro extremo, a rede neural evolutiva em cascata possui um grau de variabilidade máximo para os pesos sinápticos, porém sua estrutura topológica cresce em apenas um neurônio por iteração, e caso esse único neurônio não aumente a qualidade da solução, o neurônio é descartado e a respectiva entrada testada não será mais utilizada, causando que a solução fique presa em um mínimo local.

Os problemas dos métodos até então apresentados abre espaço para a pesquisa de um método que não falhe nas situações descritas. Ou seja, deve possuir um grau de variabilidade máximo, mesmo que isso aumente o custo computacional, almejando atingir o ótimo global; além disso, devem permitir grandes variações da topologia da rede neural, criando múltiplos novos neurônios tendo como conexões várias combinações das entradas da rede neural.

A principal característica do método de Recozimento Simulado é que ele possui uma alta variabilidade no início do treinamento, tentando aleatoriamente alcançar uma solução inicial razoável, e gradualmente diminui a variabilidade, que nesse caso é chamada de temperatura, e sua a diminuição é chamada de taxa de resfriamento, assim diminuindo o espaço de busca da rede neural e tendendo a convergir para um ótimo global. O ajuste dinâmico da taxa de variabilidade é uma grande vantagem, podendo ser ajustado pela análise estatística das soluções encontradas a cada iteração do algoritmo, e assim chegar a uma solução ótima com um tempo de processamento otimizado [Padersen 2010]. Normalmente o método de Recozimento Simulado é aplicado para topologias fixas, onde o número de neurônio é pré-estabelecido, e todos os pesos sinápticos passam pelo processo de treinamento simultaneamente, causando um espaço de busca muitas vezes impraticável quando o número de neurônio é muito alto.

A proposta desse trabalho, finalmente, é utilizar um algoritmo híbrido de recozimento simulado, porém iniciar com o número mínimo de neurônios e aumentar a cada iteração, similarmente ao método de redes neurais evolutivas em cascata. Nas próximas seções são apresentados detalhamentos da proposta.

3.1. Arquitetura da Rede Neural e da Busca por Soluções

Será tomada uma abordagem parecida a das redes neurais evolutivas em cascata apresentadas na seção 2. Na figura 3 é mostrada a sequencia de busca de soluções, partindo de soluções simples (a partir de um único neurônio), com diferentes combinações das entradas (A e B), e incrementando gradualmente o número de neurônios para buscar soluções mais complexas.

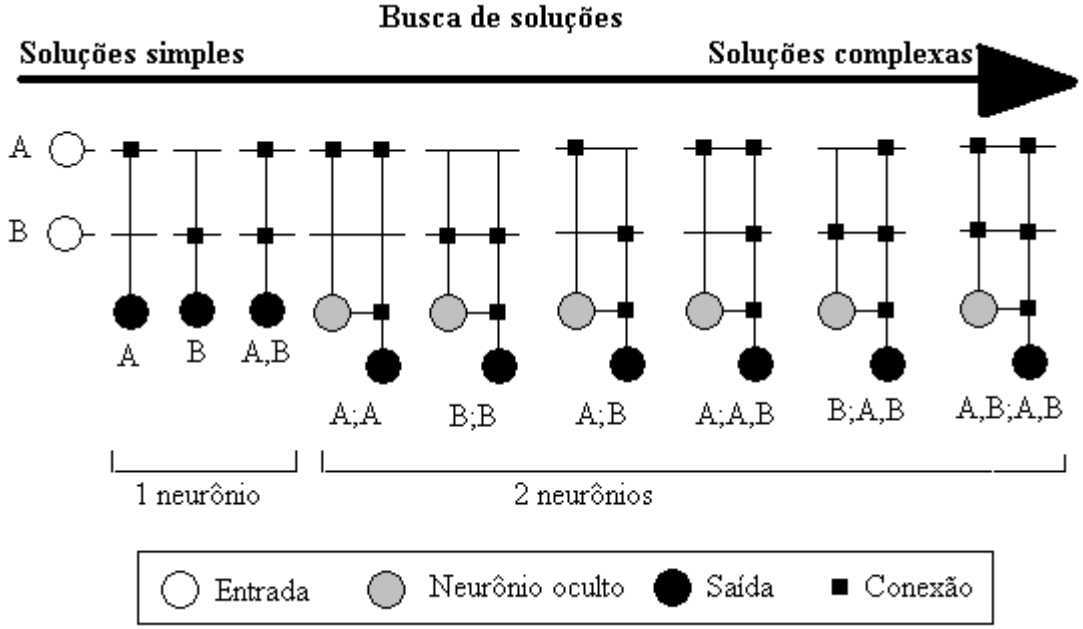


Figura 6. Sequência de detecção de soluções partindo de uma solução simples (ilustrada a esquerda) até uma solução mais complexa (ilustrada a direita). A explicação se segue. Os *neurônios de entrada* (*A* e *B)* são como sensores que obtém valores do ambiente externo. As *conexões* são como sinapses dos *neurônios de entrada* até *neurônios ocultos* ou *neurônios de saída,* ou sinapses dos *neurônios ocultos* até os *neurônios de saída.* A rede neural hipotética aqui ilustrada contém apenas um único *neurônio de saída*, mas usualmente em casos reais pode conter mais. *Neurônios ocultos* e *neurônios de saída* implementam uma *função transferência* tendo como parâmetros os valores conectados ao neurônio através das *conexões*. As *conexões* usualmente possuem um *peso,* que é um fator multiplicativo para a sua entrada. A primeira tentativa de obter uma solução é ilustrada a esquerda, contendo um único *neurônio de saída* conectado a entrada *A*, ou seja, presume-se que a saída ótima será obtida apenas com uma única função de transferência dependendo da entrada *A,* e então o neurônio é treinado pelo método de recozimento simulado. Em seguida, ilustrada imediatamente a direita da solução anterior, é testado um único neurônio conectado a entrada *B.* Em seguida é testado um neurônio contendo duas conexões, tanto para a entrada *A* quanto para a entrada *B,* supondo assim que a saída deva depender de uma combinação linear das duas entradas. Caso uma solução significativa seja encontrada nos três casos anteriores, pode-se continuar ou não a busca, dependendo de parâmetros estatísticos pré-estabelecidos que indicam se a solução encontrada obedece a um critério desejado. Caso a busca continue, o próximo passo é testar novamente diferentes combinações de entradas, mas dessa vez contendo dois *neurônios*, um *oculto*, e outro de saída. Quando há mais de um neurônio, cada neurônio acrescentado é conectado também aos *neurônios ocultos* anteriores. A busca pode continuar até um número pré-determinado de neurônios.

3.2. Algoritmo de Aprendizado

Um pseudoalgoritmo de treinamento é proposto. Para cada iteração, realiza-se a busca proposta na seção 3.1. Após ser obtida uma amostra representativa das soluções, dois passos são executados: o primeiro passo consiste em agrupar as soluções parecidas, dando origem a uma única rede neural com a temperatura dos parâmetros neuronais (pesos e limiares) ajustados de acordo com a variabilidade dos parâmetros nas soluções agrupadas, analisando-se estatisticamente os valores; o segundo passo consiste em separar soluções distintas, dando origem a novas redes neurais na população de soluções. Após esses passos, inicia-se uma nova iteração na esperança de refinar a qualidade das soluções até então encontradas.

```
// Variáveis
ENTRADAS // Entradas da rede neural
SOLUÇÕES // População de soluções encontradas
NN // Número de neurônios

// Variáveis pré-estabelecidas
MNN // Número máximo de neurônios

// Algoritmo de treinamento
SOLUÇÕES ← solução inicial aleatória com máximo grau de variabilidade
FAÇA, paralelamente para as SOLUÇÕES até então encontradas
        FAÇA
                PARA (NN=1 até MNN)
                        PARA (diferentes combinações de entradas)
                                Adicionar NN neurônios
                                Testar soluções
                                Salvar soluções encontradas em SOLUÇÕES
                        FIM
                FIM
                Realizar cálculo estatístico das SOLUÇÕES
        ATÉ QUE (seja encontrada uma amostra representativa do problema)
        Medir diferenças entre redes neurais em SOLUÇÕES
        Agrupar soluções parecidas, calculando a variação máxima dos pesos e limiares (temperatura)
        Separar soluções distintas, recolocando-as em SOLUÇÕES
        Realizar cálculo estatístico das SOLUÇÕES
ATÉ QUE (seja encontrada uma amostra representativa do problema)
```

4. Definição das Funções a Serem Aproximadas

O principal objetivo dessa proposta é obter soluções em ambientes complexos. Para validar cientificamente essa capacidade, é proposta uma adaptação do teste de aproximação de funções.

Para simplificação, as funções a serem aproximadas e as funções de transferência utilizadas nos neurônios serão as mesmas. Assim, pode-se avaliar se a rede neural foi capaz de encontrar a solução com o número mínimo de neurônios ocultos e de conexões, pois assim o número de neurônios deverá corresponder às componentes da função a ser aproximada, e o número de conexões deverá corresponder aos parâmetros do qual a função a ser aproximada depende. A figura 7 e 8 mostram um exemplo de

função a ser aproximada, bem como a respectiva rede neural mínima que é capaz de representar a função.

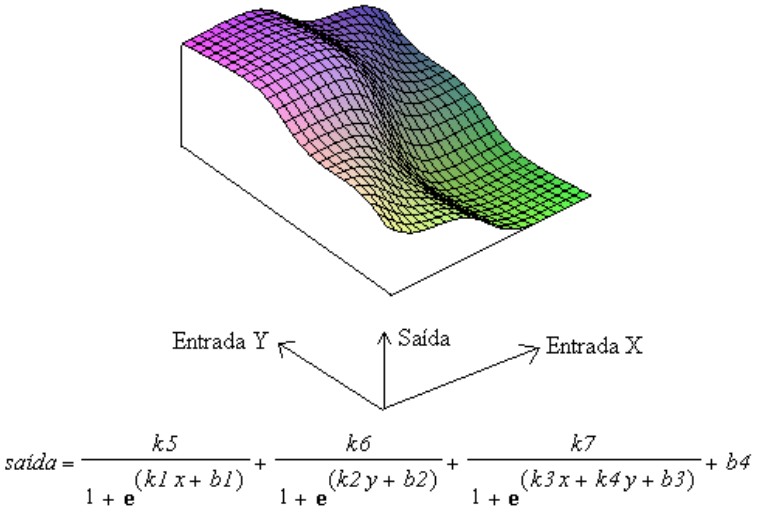


Figura 7. Exemplo de função a ser aproximada contendo a soma de três componentes sigmoides. A primeira sigmoide (primeiro termo da soma) depende unicamente da entrada *X* multiplicada por um fator *k1* somada com um limiar *b1.* A segunda sigmoide depende unicamente da entrada *Y* multiplicada por um fator *k2* somada com um limiar *b2.* A terceira sigmoide depende de ambas as entradas *X* e *Y,* cada qual multiplicada por um fator *k3* e *k4,* e somado com um limiar *b3.* As sigmoides são multiplicadas por constantes *k5, k6* e *k7* e finalmente somadas com o limiar *b4.*

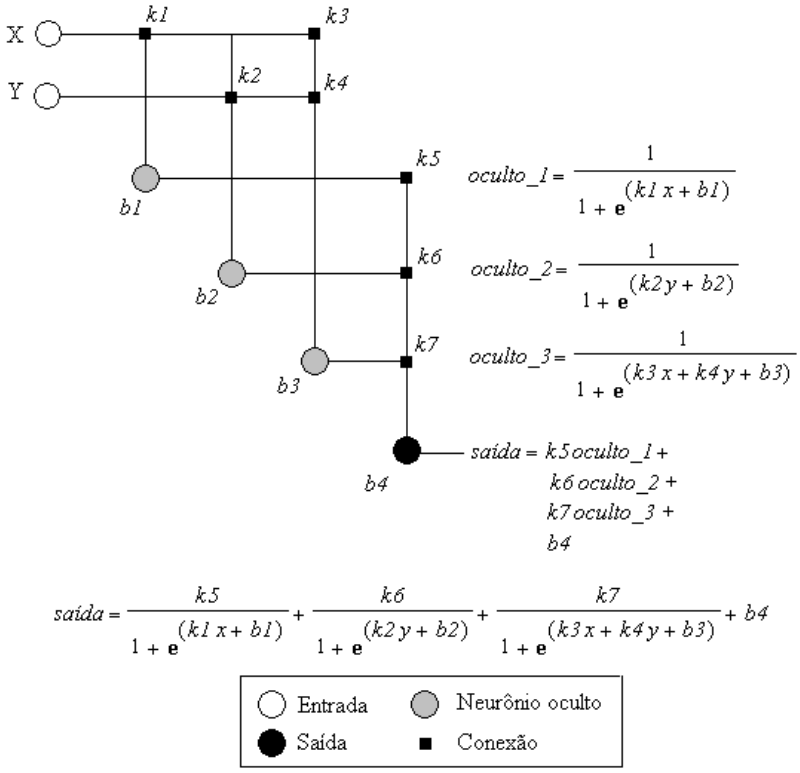


Figura 8. Rede neural mínima a ser obtida para representação da função a ser aproximada ilustrada na figura 7. A função de saída da rede neural é idêntica à função a ser aproximada.

O gráfico da figura 9 ilustra o espaço de estados da rede neural em relação à qualidade da solução que deve ser simulado por três soluções a ser aproximadas, representando as múltiplas soluções a serem obtidas pelo algoritmo de treinamento.

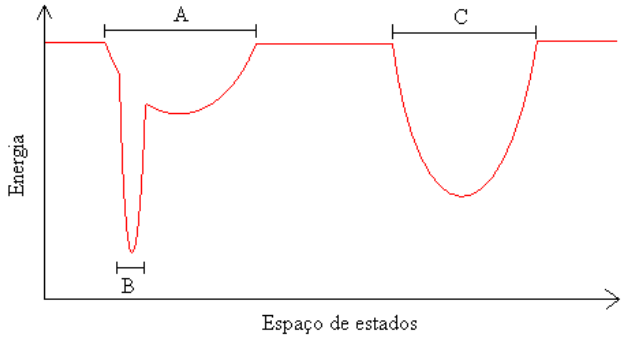


Figura 9. Exemplo do espaço de estados a ser simulado. As soluções *A* e *C* são distintas, sendo simuladas por duas funções com pesos e limiares completamente diferentes. É esperado do algoritmo de treinamento identificar ambas as soluções em uma primeira iteração. A solução *B* representa uma especialização da solução *A*, contendo componentes adicionais em relação a *A*, porém complexa e específica demais (representação estreita no gráfico) para ser encontrada com alta probabilidade em uma primeira iteração, devendo ser encontrada em uma segunda iteração do algoritmo. O espaço de estados deve conter regiões onde a qualidade da solução é nula (energia máxima), simulando o efeito caixa-preta de incerteza, e o algoritmo de aprendizado deve ignorar tais soluções encontradas.

As soluções mais largas *A* e *C* representadas na figura 9 devem ser mais facilmente encontradas pelo algoritmo de treinamento, assim a qualidade da solução admite maior variabilidade. Já a solução *B* é representada de forma estreita no gráfico, o que significa que o algoritmo de treinamento terá maior dificuldade em encontrar os pesos e limiares para representá-la. Além disso, a solução *B* é a de melhor qualidade, e assim que a rede neural conseguir encontrá-la, deverá terminar o algoritmo de treinamento.

5. Planejamento de Atividades

Tabela 1: Plano de atividades

Tarefa	*Horas*	*Risco*	*Comple-xidade*	*Importância*	*Ordem*
Arquitetura básica da rede neural	20	Baixo	Alta	Servirá de base para desenvolver o algoritmo de treinamento.	1º
Interface de criação de novos neurônios	10	Baixo	Média	Servirá de base para desenvolver o algoritmo de treinamento.	2º
Interface de medida de diferença entre redes neurais	10	Alto	Baixo	Métrica importante para avaliar diferentes soluções encontradas pelo algoritmo de treinamento.	3º
Funções a serem aproximadas (Black Box)	5	Baixo	Baixa	Principal parâmetro para a validação do algoritmo, devendo simular ambientes complexos.	4º
Algoritmo de treinamento	10	Baixo	Alto	Parte do projeto onde ainda poderão ser feitas pesquisas e melhoras.	5º
Estatística	20	Alto	Alto	Deve-se buscar implementações prontas, ou em outro caso, implementá-las.	5º
Teste, possíveis aprimoramentos, conclusões e monografia	25	Baixo	Médio	Boa formulação das conclusões e futuro trabalho.	6º
Total	100				

6. Conclusão

O problema de aprendizado de máquina tem se mostrado um problema de difícil solução, com diversas adaptações para situações específicas. A adaptação de técnicas tornou-se um procedimento padrão, exigindo vários ajustes em parâmetros de algoritmos e muitas vezes não se conseguindo alcançar soluções aceitáveis.

Com este projeto espera-se obter um método eficiente de aprendizado nos seguintes requisitos: alcançar a solução ótima com alta probabilidade e precisão; alcançar a solução ótima minimizando o tamanho da rede neural; minimizar o número de parâmetros ajustáveis pré-escolhidos; alcançar a solução ótima com desempenho aceitável (alta eficiência computacional); alcançar a solução ótima sistematicamente independente da complexidade da rede neural exigida para a solução.

O algoritmo de treinamento de redes neurais proposto utiliza um misto de técnicas já amplamente utilizadas e validadas no corrente estado da arte. Algumas modificações nos algoritmos originais foram apresentadas, aumentando a complexidade do projeto para a integração dos métodos escolhidos.

Referências

Basso, D. M. (2005) “Arquitetura Neural para Controle da Locomoção de um Robô Quadrúpede Baseada em Referências Biológicas”. 2005. 54 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

Black Box. Wikipedia. Disponível em <http://en.wikipedia.org/wiki/Black_box>. Acesso em: jun. 2010.

Elman, J. L. (1993) “Learning and development in neural networks: the importance of starting small” *Cognition, 48.* Departments of Cognitive Science and Linguistics, University of California, San Diego, La Jolla, CA 92093-0526, USA.

Fahlman, S. E. e Lebiere, C. (1991) “The Cascade-Correlation Learning Architecture.” School of Computer Science, Carnegie Mellon University, Pittsburgh.

Interactive Statistical Calculation Pages. Disponível em <http://statpages.org>. Acesso em: jun. 2010.

Kohl, N. (2009) “Learning in Fractured Problems with Constructive Neural Network Algorithms”. Thesis – (Doctorate in Philosophy) – Faculty of the Graduate School, The University of Texas at Austin.

Neyman, A. (2003) “Algorithmic Strategy Complexity”. Hebrew University of Jerusalem, Jerusalem Israel. Disponível em: <http://ratio.huji.ac.il/dp/neyman/algorslidesnwu.pdf>. Acesso em: jun. 2010.

Padersen, M. E. H. (2010) “Tuning & Simplifying Heuristical Optimization”. Thesis – (Doctorate in Philosophy) – School of Engineering Sciences, University of Southampton.

Ram, D. J., Sreenivas, T. H., Subramaniam, K. G. (1996) “Parallel Simulated Annealing Algorithms”. *Journal of Parallel and Distributed Computing* 37, n. 121, p. 207–212.

Root Mean Square Deviation. Wikipedia. Disponível em <http://en.wikipedia.org/wiki/Root_mean_square_deviation>. Acesso em: jun. 2010.

Schetinin, V. A. “Learning Algorithm for Evolving Cascade Neural Networks”. *Neural Processing Letters* 17, p. 21–31, 2003.

Stanley, K. O., Miikkulainen, R. (2001) “Evolving Neural Networks through Augmenting Topologies” Department of Computer Sciences, The University of Texas at Austin, Austin, TX.

Stanley, K. O., Bryiant B. D., Miikkulainen, R. (2005) “Evolving Neural Network Agents in the NERO Video Game”. *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games.*

Universal Approximation Theorem. Wikipedia. Disponível em <http://en.wikipedia.org/wiki/Universal_approximation_theorem>. Acesso em: jul. 2010.

APÊNDICE – TABELAS DE RESULTADOS

Segue uma breve explicação das tabelas:

(ρ) refere-se à parcela da distribuição de probabilidades que deve ser aceita como amostra, conforme a seção 3.3 explica;

(1/2/4/8), no eixo vertical, refere-se ao número mínimo de amostras necessário para realizar uma nova *clusterização* de amostras, conforme a seção 3.4.

(x $\bar{u}$, x σ): x refere-se à confiança em desvios-padrão da medida que o algoritmo de *clusterização* utiliza para unir dois *clusters* (ou uma rede neural e um *cluster*), conforme a seção 3.4.

($\bar{u}$ e σ) é a média e desvio-padrão do número de execuções da simulação física do duplo carro-pêndulo até obtenção da solução.

(falhas / N) refere-se ao número de falhas (o algoritmo não encontrou uma solução) em relação ao número total de execuções do algoritmo *(N)*.

Tabela 1 – Resultados do algoritmo no problema de carro-pêndulo[1]

		1,2 $\bar{u}$	**1,2 σ**	**1,4 $\bar{u}$**	**1,4 σ**	**1,6 $\bar{u}$**	**1,6 σ**	**1,8 $\bar{u}$**	**1,8 σ**	**2,2 $\bar{u}$**	**2,2 σ**
ρ=0,2	**1**			10342	6816	10175	5273	9273	6149	8912	5833
ρ=0,15	**1**	8968	4350	5362	2705	5989	3175	5823	2902	5735	3151
ρ=0,1	**1**	7882	5905	6569	8979	4386	3297	3699	2142	4230	2633
ρ=0,05	**1**							9531	16291		

1 O número de amostras foi ajustado para obter um erro máximo de ±20% na média, com confiança de 95%, assumindo uma distribuição normal.

Tabela 2 – Resultados do algoritmo no problema de duplo carro-pêndulo[2]

		2,0 $\bar{u}$	**2,0 σ**	**2,4 $\bar{u}$**	**2,4 σ**	**2,8 $\bar{u}$**	**2,8 σ**	**3,2 $\bar{u}$**	**3,2 σ**
ρ=0,15	**1**			2642	1508	2465	2048	2422	1307
ρ=0,15	**2**			2226	1419	2336	1654	2308	1216
ρ=0,15	**4**			2537	2558	2581	1912	2275	2114
ρ=0,15	**8**			3145	4185	2727	2457	2783	2523

		2,0 $\bar{u}$	**2,0 σ**	**2,4 $\bar{u}$**	**2,4 σ**	**2,8 $\bar{u}$**	**2,8 σ**	**3,2 $\bar{u}$**	**3,2 σ**
ρ=0,1	**1**	2083	1279	2028	1386	1824	1213	1962	1205
ρ=0,1	**2**	2166	1308	2160	1367	2038	1402	2136	1231
ρ=0,1	**4**	2927	3750	2540	2836	2194	1626	2361	2130
ρ=0,1	**8**			2671	2889	2668	2628	2494	1920

				2,4 $\bar{u}$	**2,4 σ**	**2,8 $\bar{u}$**	**2,8 σ**	**3,2 $\bar{u}$**	**3,2 σ**
ρ=0,05	**1**			2085	1707	1931	1188	2026	1828
ρ=0,05	**2**			2278	2181	1954	1284	1978	1194
ρ=0,05	**4**			2721	2894	2310	1796	2116	1703
ρ=0,05	**8**			3022	3279	3027	3816	2539	2467

2 O número de amostras foi ajustado para obter um erro máximo de ±10% na média, com confiança de 95%, assumindo uma distribuição normal.