chemkeys®
Liberdade para aprender

doi: 10.20396/chemkeys.v3i00.15880

Redes Neurais para Regressão Uni- e Multivariada

Gabriel César Pereira
g212083@dac.unicamp.br

*Rogério Custodio**
rogerct@unicamp.br
Universidade Estadual de Campinas, Instituto de Química

Informações sobre o artigo

Histórico do Artigo

Submetido em 01 de julho de 2021
Aceito em 16 de agosto de 2021

Palavras-chave:
Redes Neurais Artificiais
Regressão
Algoritmo Genético
Recozimento Simulado
Retropropagação
Otimização

Resumo

Redes Neurais Artificiais têm ganhado notoriedade na aproximação de funções uni e multivariadas em virtude a alta capacidade aproximativa desse tipo de modelo. Neste artigo é apresentada uma descrição dos modelos de regressão baseados em redes neurais juntamente com os algoritmos comumente usados para otimizá-los. A performance deste tipo de modelo é exemplificada através da aproximação de uma função univariada que relaciona a fração em mol na fase líquida de um dos componentes de uma mistura água-acetona com sua fração em mol na fase de vapor. O desempenho do modelo é, ainda, comparado com o desempenho de outros modelos baseados em métodos de regressão clássicos utilizados para solucionar o mesmo problema. Ao final do texto, é apresentado o código PYTHON para a criação do modelo de rede neural discutido aqui.

(cc) BY-NC-SA

Introdução

Uma análise uni- ou multivariada tem por objetivo principal quantificar uma determinada propriedade de interesse utilizando modelos desenvolvidos a partir de dados que possuam alguma suposta correlação, mesmo que esta seja, à priori, desconhecida. Em diversas áreas do conhecimento a modelagem de dados é feita de forma corriqueira com o propósito de associar variáveis independentes, tais como concentrações, temperaturas e volumes, com respostas como velocidades de reações, absorbância, energia de reação.

Dentro do conjunto de métodos e algoritmos disponíveis para a regressão de dados, um dos mais populares, por sua simplicidade e excelente desempenho, é o *Método dos Mínimos-Quadrados*. Este método permite modelar matematicamente uma curva minimizando a soma dos quadrados dos resíduos definidos pela diferença *valor de referência – valor aproximado* [1]. A equação final da curva ajustada é definida pelo conjunto de parâmetros que melhor ajusta quadraticamente os dados aproximados dos de referência, podendo assumir diferentes modelos como o linear, o polinomial, o logarítmico, o exponencial etc.

$y_i = \beta_0 + \beta_1 x_i$	modelo linear
$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots + \beta_m x_i^m$ (i=1, 2,., n)	modelo polinomial
$y_i = \beta_0 + \beta_1 \ln(x_i)$ em que $x_i \geq 0$	modelo logarítmico
$y_i = \beta_0 \beta_1^{x_i}$ em que $\beta_0 \neq 0$	modelo exponencial

Um aspecto importante em comum nos modelos acima é que eles permitem descrever de maneira clara a relação entre as variáveis independentes-resposta através da equação da curva ajustada pelos parâmetros beta. Isto é importante e necessário em casos em que se deseja realizar uma análise mais rigorosa de como cada termo do modelo matemático contribui para a descrição da referida relação informação-resposta.

Assim como qualquer modelo, os exemplos acima apresentam limitações. Elas são inerentes à simplicidade matemática que não permite, ocasionalmente, descrever com alto rigor preditivo as relações demasiadamente complexas entre as variáveis. Considere por exemplo uma solução não-ideal de acetona e água para a qual se deseja ajustar a fração em mol da fase líquida x_1 de um dos componentes, a acetona por exemplo, em relação à sua fração em mol na fase de vapor y_1 [2]. Observando as regressões realizadas na Figura 1 constata-se que ajustes polinomial e logarítmico são limitados para descrever a relação existente entre essas duas variáveis.

Em casos como este, onde há margem para implementação de um modelo melhor otimizado para dados de difícil regressão, os métodos baseados em *machine learning* ou *aprendizado de máquina* se mostram geralmente mais eficientes. O termo *machine learning* é utilizado para se referir aos algoritmos capazes de "aprender" ou de "se auto otimizar" a partir dos dados de treinamento (ou da "experiência"). Dentre esses métodos, o das *Redes*

*Autor para contato

Neurais Artificiais é considerado um dos mais populares, devido principalmente à sua capacidade de aproximar universalmente qualquer função a qualquer conjunto de dados [3-11].
Redes neurais artificiais são modelos computacionais ins-

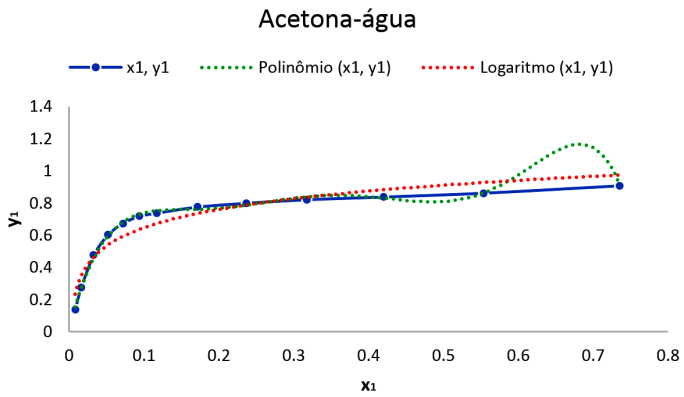
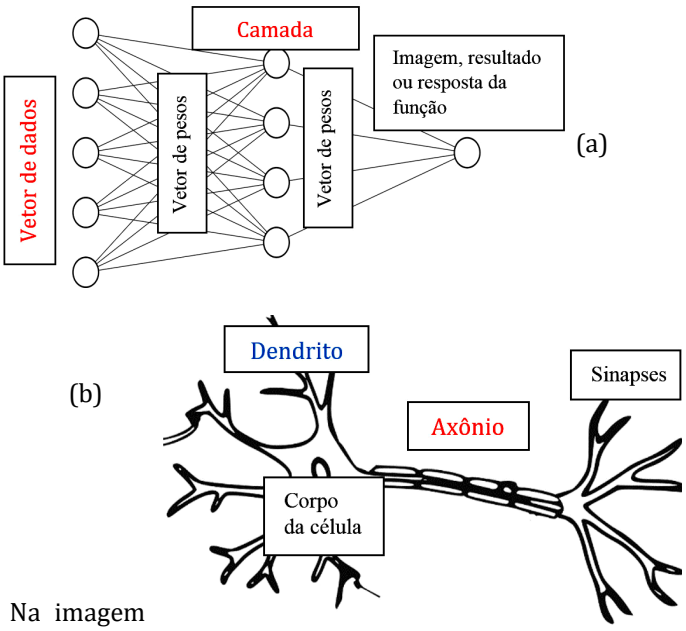


Figura 1 – Ajuste polinomial:
$Y=-1080,2x^6+2373,7x^5-2007,8x^4+828,8x^3-174,8x^2+18,15x+0,021$;
Ajuste logarítmico:
$Y= 0,1639 \ln(x)+1,025$;
Erro médio absoluto de 0,011 e 0,054 respectivamente.

Redes neurais artificiais

pirados no sistema nervoso de seres vivos [12]. Assim, essas redes podem ser definidas como um conjunto de unidades de processamento, caracterizadas por *neurônios artificiais* ou *"nodes"* (*nós*) que são interligados por diversas interconexões (*sinapses artificiais*), sendo representadas matematicamente por vetores ou matrizes de pesos sinápticos. Essas interconexões são prolongadas e reprogramadas através de camadas dentro da estrutura da rede:



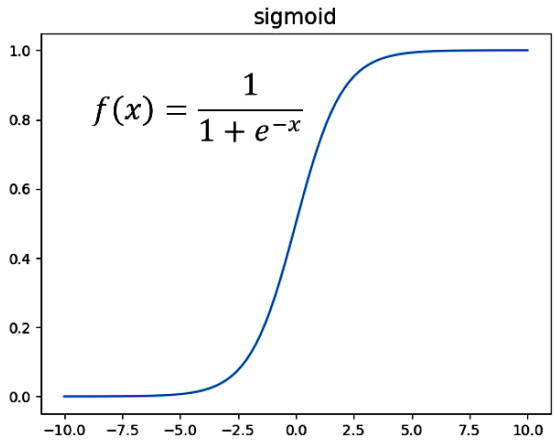
Na imagem acima podemos observar a correspondência entre uma rede neural artificial e um neurônio biológico. Em uma rede neural, cada unidade de processamento ou círculo pertencente à(s) camada(s) interna(s) é considerado um neurônio. Cada neurônio biologicamente recebe a informação através dos dendritos, processa a informação recebida no corpo da célula e a envia através do axônio e das sinapses. A sua associação com a rede neural artificial ocorre a partir de uma combinação linear (vetor de pesos) de um conjunto de dados iniciais (vetor de dados), que é transferida para cada neurônio da camada interna. Cada neurônio artificial transforma os valores da combinação linear recebidos por meio de uma função matemática denominada *função de ativação*, que avalia como a informação prossegue para a camada seguinte. As respostas de cada neurônio provenientes da função de ativação são combinadas linearmente empregando novamente um vetor de pesos e é transferida para os demais neurônios da camada seguinte. É possível notar até aqui que o modelo acima, apesar de simples, pode assumir qualquer nível de complexidade desejada, dependendo do número de unidades de processamento, *ou seja*, do *número de neurônios* e *camadas internas* que se queira utilizar. De fato, uma rede contendo duas camadas, por exemplo, pode aproximar qualquer função contínua dentro de um domínio relativamente compacto com qualquer valor arbitrário de exatidão, desde que a estrutura da rede possua unidades internas (neurônios e camadas internas) suficientes.

Figura 2 – (a) modelo de uma rede neural do tipo que não opera em ciclos (*feed forward)*; (b) ilustração de um único neurônio orientado de maneira correspondente ao modelo.

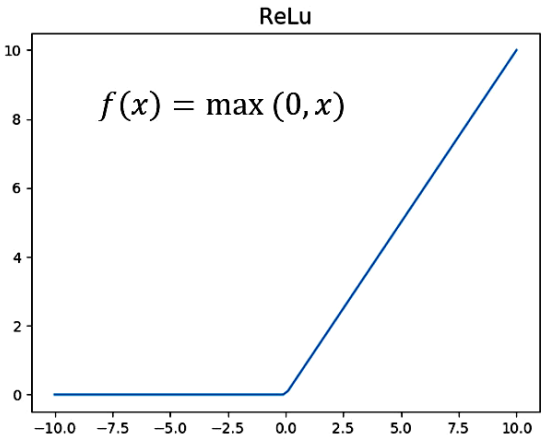
Apesar de inicialmente ter se tornado popular como um algoritmo de classificação, as redes neurais artificiais também são capazes de realizar regressão uni- ou multivariada, sendo que esta última demonstra o seu potencial sobre os demais métodos devido à sua capacidade de manipular adequadamente grandes volumes de dados. Portanto, a vantagem da rede neural artificial como método de regressão em comparação com os métodos acima está na sua capacidade autorregulatória e na possibilidade de lidar satisfatoriamente com condições não-lineares. A capacidade autorregulatória se refere ao poder de adaptação e otimização dos vetores de pesos sinápticos durante o treinamento do modelo. Esse aspecto torna-se notável principalmente considerando que um modelo de rede neural clássico pode conter milhares de pesos distribuídos ao longo das camadas internas a depender do nível de rigor pretendido na previsão ou dificuldade de aproximar uma solução para um determinado problema. Já o caráter não-linear advém das funções de ativação que transformam os valores recebidos dos neurônios através do processamento de cada um deles por uma função usualmente não-linear, podendo assumir diversas formas: *Sigmoid*, *ReLu(unidade linear retificada)*, *tanh (tangente hiperbólica)*, como mostra a Figura 3, entre outras.

A escolha pela melhor função de ativação a ser aplicada em cada camada está sujeita ao perfil de dados do problema [13]. Se o objetivo do modelo é realizar uma classificação binária, por exemplo, a função *sigmoid* é uma boa opção uma vez que resulta apenas em valores positivos entre 0 e 1. Se o objetivo, no entanto é realizar uma regressão cujos valores de saída são todos maiores que 0,

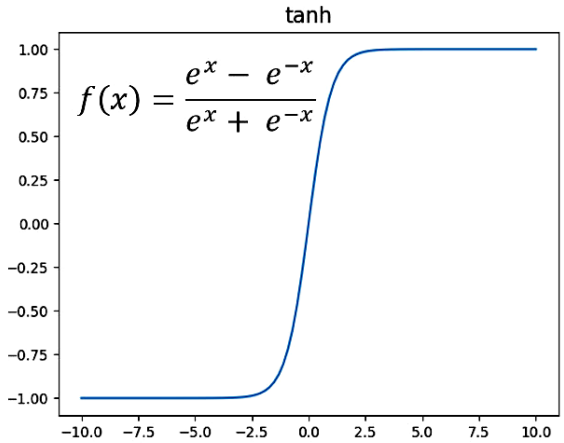
a função ReLu pode ser a melhor opção, enquanto para valores menores ou maiores que 0 a função *tanh* poderá desempenhar melhor.



(a)



(b)



(c)

Figura 3 – Exemplos de funções de ativação empregadas sobre as camadas em uma rede neural. Cada valor de x recebido pelo neurônio é transformado e repassado adiante na

[1] No contexto do algoritmo genético, o termo *fitness* generaliza a ideia de performance (*i.e.* menor, valor da função erro) para problemas de maximização e minimização de funções.

Matematicamente pode-se representar um modelo de regressão baseado em rede neural artificial da seguinte forma:

$$E(w_1,\ldots,w_M) = \sum_{n=1}^{N} [z_n - y_n(w_1, \ldots, w_M)]^2$$

O conjunto de parâmetros w corresponde aos pesos sinápticos que serão otimizados para minimizar o erro $E(w_1, \ldots, w_M)$ calculado como a soma do quadrado das diferenças entre o valor experimental ou de referência da amostra z_n e o respectivo valor calculado com a função $y_n(w_1, \ldots, w_M)$. A formulação genérica apresentada para a função modelo $y_n(w_1, \ldots, w_M)$ contempla também as possíveis funções de ativação empregadas no processamento dos dados, a ser exemplificado mais adiante. Por fim, assim como todo modelo de previsão, o erro obtido será o indicativo do nível de otimização de seus parâmetros ou até mesmo das limitações do modelo. Com isso, existem no contexto das redes neurais diversos algoritmos disponíveis para a otimização dos parâmetros ou pesos w, alguns exemplos são: *Retropropagação* (*Backpropagation*) [14], *Algoritmo Genético* (Genetic Algorithm - GA)[15], e *Recozimento Simulado* (*Simulated Annealing*)[16].

Algoritmo genético

O algoritmo genético é um procedimento empregado na busca ou otimização de processos e utiliza a estratégia empregada por processos evolutivos [17] cuja principal característica é a rapidez com que estes mudam com o tempo, distinguindo-os dos algoritmos estáticos tradicionais. Dada uma função objetiva, ou no contexto das redes neurais uma *função erro*, o vetor ou conjunto de parâmetros ótimos que minimiza essa função é obtido iterativamente por meio da combinação ou cruzamento dos conjuntos de parâmetros N com melhor desempenho (maior *fitness*[1]) ao final da computação da função erro. Em termos práticos:

1– O algoritmo é inicializado com a geração de conjuntos de parâmetros que permitirão o cálculo do valor da função erro.

2- Realiza-se o cruzamento em que os valores para cada parâmetro são reorganizados ou combinados para dar origem ao mesmo número de parâmetros, ou genes neste contexto, produzindo o que se denomina de próxima geração.

3- O processo se repete a partir dos indivíduos da nova geração que agora são considerados como população inicial.

Três aspectos são centrais na implementação das três etapas apresentadas acima. São eles: o método de escolha dos membros de cada geração a realizar o cruzamento, a escolha do método de cruzamento e o critério de convergência. Existem diversas formas de realizar a seleção dos membros de cada geração. Algumas dessas formas são tão intuitivas quanto "selecionar apenas os membros com melhor resultado" enquanto outras permitem levar em

conta o espaço de possibilidades de forma mais flexível. Alguns dos métodos de seleção de membros para o cruzamento[2] estão listados a seguir juntamente com uma breve descrição de cada um em sua respectiva nota de rodapé: torneio[3], roleta[4] e elitista[5].

Sabe-se que cada membro representa um conjunto de parâmetros que, no contexto das redes neurais, corresponde aos parâmetros ou pesos dessa rede. Podemos então considerar d = [w_1, w_1, w_1, ... w_n] como um membro "descendente" com parâmetros w_1, w_1, w_1, ... w_n oriundos do cruzamento de dois membros (parentes) da geração anterior. Para tanto, é preciso definir de que maneira serão combinados os parâmetros dos membros parentes para obter os parâmetros de d. A seguir estão listadas algumas formas conhecidas de se realizar esse cruzamento e, na Tabela 1, estão contidas descrições de cada um desses métodos: cruzamento "achatado" (Flat crossover) [18], cruzamento simples (Simple crossover) [19], cruzamento aritmético (Arithmetic crossover) [20], cruzamento discreto (Discrete crossover) [21] e cruzamento BLX-α [22]. Os subscritos e sobrescritos na tabela abaixo correspondem respectivamente ao parâmetro e ao parente ou descendente, e d_i corresponde ao gene ou parâmetro i do membro descendente d.

Tabela 1 – Diferentes formas de calcular cada um dos parâmetros do vetor descendente.

Crossover achatado	$d_i=w_i^1+r_i(w_i^2-w_i^1)$, se $w_i^1-w_i^2$	Sendo $0<r_i<1$ obtido aleatoriamente
Crossover simples	$d^1=(w_1^1,...,w_i^1, w_{(i+1)}^2,...,w_n^2)$ $d^2=(w_1^2,...,w_i^2, w_{(i+1)}^1,...,w_n^1)$	Sendo $i \in \{1,2,...n-1\}$ obtido aleatoriamente para gerar dois descendentes
Crossover aritmético	$d_i^1=rw_i^1+(1-r)w_i^2$ $d_i^2=rw_i^2+(1-r)w_i^1$	Sendo $-0,5\leq r \leq 0,5$ obtido aleatoriamente para gerar dois descendentes
Crossover discreto	$d_i=w_i^1$ ou $d_i=w_i^2$	d_i é aleatoriamente escolhido a partir do conjunto $\{w_i^1, w_i^1\}$
BLX-α	$w_{(min)}-\alpha I \leq d_i \leq w_{(max)}+\alpha I$	d_i é aleatoriamente obtido a partir do intervalo à esquerda, onde $w_{(max)} = max(w_i^1, w_i^2)$, $w_{(min)} = min(w_i^1, w_i^2)$, $I = w_{(max)} - w_{(min)}$ e $\alpha = 0,5$ ou outro valor otimizado empiricamente.

Para exemplificar, considere a busca pelo mínimo da função representada pelo polinômio: $y = 3w_1^2 - 6w_2 + w_2$ sendo w_1, w_2, w_3 os parâmetros a serem determinados. A inicialização do processo de otimização ocorre com a escolha de valores aleatórios para os parâmetros w em uma pequena população com 4 membros (número de membros escolhido arbitrariamente). Temos aqui, então, 4 conjuntos de parâmetros [w_1, w_2, w_3] possíveis. Substituindo os parâmetros do polinômio acima pelos valores dos parâmetros de um desses membros, obtermos o seu respectivo valor de y, ou no contexto do algoritmo genético, o seu ajuste (*fitness*). Aqui é importante reforçar que o ajuste de um membro é a medida do quão bom é o seu desempenho, de modo que a partir desse valor o usuário possa ranquear os membros e então selecionar para o cruzamento aqueles com as melhores performances. É fácil ver, assim, que os membros posicionados no topo do ranking podem ser os que apresentarem o maior ou menor valor de y a depender do problema em questão, isto é, maximização ou minimização da função respectivamente.

Como proposto acima, considere neste momento que desejamos minimizar a função polinomial dada e, para tal, calculamos o valor do ajuste para cada um dos 4 membros gerados acima para a inicialização do procedimento. Utilizando o método de seleção elitista (referir a nota de rodapé anterior) e o método de cruzamento discreto (veja a Tabela 1)[6] cria-se uma população a partir dos descendentes desses 4 membros (Tabela 2). Aqui, a *nova população* irá considerar também o ajuste da geração ou população anterior na classificação final dos ajustes (do melhor ao pior).

Em problemas práticos a população inicial é gerada com dezenas ou até mesmo centenas de indivíduos. Os indivíduos que apresentarem melhor resultado no tocante à função erro são utilizados na etapa (2) de cruzamento, sendo seus respectivos valores para cada parâmetro são combinados para dar origem à próxima geração. Um conhecimento prévio das características do problema em questão pode auxiliar na escolha dos métodos de seleção e cruzamento mais apropriados. Mas estes são mais seguramente identificados por meio de tentativa e erro.

Um outro elemento importante no AG é a *mutação*, cuja finalidade é prover uma fonte de variabilidade aos parâmetros de modo a evitar mínimos locais [15]. A mutação ocorre com uma probabilidade pré-estabelecida, sendo importante notar que se a probabilidade for muito pequena, a mutação raramente acontece e o algoritmo converge mais rapidamente sem, consequentemente, explorar o mínimo global. Por outro lado, se a probabilidade for muito grande, o algoritmo levará muito mais tempo para convergir, podendo conferir um custo computacional muito maior. Alguns métodos de mutação são apresentados na Tabela 3. Finalmente, a convergência é considerada obtida quando o ajuste não continua a melhorar após um número pré-determinado de gerações; este número tende a depender do problema em questão, sendo 3 e 5 os números comumente usados.

[2] Do inglês, *crossover*.
[3] Do inglês, *Tournamen*t. Uma amostra aleatória com *N* indivíduos da população é isolada. Dentro dessa amostra, o membro com melhor *ajuste* é selecionado para o cruzamento.
[4] Do inglês, *Roulette wheel*. Cada membro *i* tem uma probabilidade de ser selecionado para o cruzamento. Essa probabilidade é uma função da performance do seu ajuste.
[5] Os membros são ordenados de acordo com seu *ajuste*. Os melhores *N* membros são selecionados para o cruzamento. Membros de gerações anteriores também são utilizados.
[6] Os métodos foram selecionados apenas como exemplos e não são necessariamente os ótimos.

Tabela 2 – Esquema simplificado do fluxo de otimização característico do Algoritmo Genético (AG). Uma população é iniciada com um número N de indivíduos. Os melhores indivíduos (de acordo com a função erro) são cruzados para dar origem a próxima geração. Os melhores indivíduos dessa nova geração são cruzados dando prosseguimento a otimização até a convergência ser obtida.

N	w_1	w_2	w_3	*fit.*		Cruz.	Cruz.	N	w_1	w_2	w_3	*fit.*		*Nova população*	x_1	x_2	x_3	*fit.*
1	-0,01	1,35	-0,42	-8,52	➡	1	2	5	-0,01	0,65	-0,42	-4,32		7	-0,08	1,7	-0,42	-10,6
2	1,47	0,65	-0,11	2,47	➡	2	3	6	-0,08	0,65	0,11	-4,00		3	-0,08	1,70	0,25	-9,93
3	-0,08	1,70	0,25	-9,93	➡	3	1	7	-0,08	1,7	-0,42	-10,6		1	-0,01	1,35	-0,42	-8,52
4	1,12	0,40	-0,91	5,25										5	-0,01	0,65	-0,42	-4,32

Tabela 3 – Métodos de mutação comumente utilizados no Algoritmo Genético (AG).

Troca de *bit*	0	0	1	1	0	1	0	0	0	1		0	0	1	1	0	1	1	0	0	1
Troca	0	1	2	3	4	5	6	7	8	9		0	6	2	3	4	5	1	7	8	9
Embaralhamento	0	1	2	3	4	5	6	7	8	9		0	1	3	6	4	2	5	7	8	9
Inversão	0	1	2	3	4	5	6	7	8	9		0	1	2	3	8	7	6	5	4	9

Recozimento simulado

O *recozimento* ou *annealing* no campo da ciência dos materiais é o termo utilizado para se referir ao processo de fundição de um metal quando este é aquecido a uma alta temperatura e, então, é resfriado lentamente para a obtenção de uma massa homogênea [16]. Nesse processo, substâncias físicas passam de um estado de alta energia para um estado de baixa energia. Analogamente, no contexto da otimização de modelos esse fenômeno equivale a diminuir o erro ou a "energia" do modelo. No caso dos materiais, um olhar microscópico sobre esses sistemas irá revelar que, em virtude da alta temperatura, estados de maior energia também podem ser ocupados por alguns átomos com uma dada probabilidade. Esse aspecto também é análogo a estabelecer uma probabilidade para que um ajuste que leve ao aumento da função erro possa ser aceito durante o processo de otimização. Essa fonte de variabilidade adicional flexibiliza os passos da busca de modo a evitar mínimos locais. Se esse "resfriamento" ou otimização ocorre de forma lenta, estudos mostram que a estrutura final tende a ser a de menor 'energia' ou erro [23].

Na prática, o *Recozimento Simulado* é utilizado sempre que se suspeita que se esteja em um mínimo local ou para confirmar que se está em um mínimo absoluto. O algoritmo é muito simples e necessita de poucos passos:

1 - Inicialização com valores aleatórios dos parâmetros a serem otimizados.

2 - Definir uma alteração ou perturbação cuja ordem de grandeza é determinada pelo usuário. Se o erro do novo passo é menor que o do anterior o passo é aceito. Porém, se o valor do erro referente ao novo passo for maior do que o valor do anterior, o algoritmo poderá permitir a permanência desse novo passo mediante uma probabilidade determinada pelo critério de Metropolis(1). Desse modo, uma função de probabilidade P comumente utilizada para definir se o passo será ou não aceito é a função do tipo Boltzmann:

$$P = e^{\frac{f(w)-f(w+passo)}{T_n}}$$

em analogia à distribuição de Boltzmann[7]: $P_i \propto e^{\frac{\epsilon_i}{kT}}$

Sendo assim, se *P* for maior que um número selecionado aleatoriamente no intervalo [0,1], o passo será aceito mesmo que sua 'energia', *f(w+passo)*, seja maior que a 'energia' do estado anterior, *f(w)*.

3 - A cada etapa n de 'resfriamento', um total de i iterações são realizadas de modo que seja alcançado o equilíbrio térmico àquela temperatura; a temperatura T_n é então reduzida por meio de um fator α de acordo com: $T_{n+1} = \alpha T_n$. Quanto menor a temperatura T menor a probabilidade de um passo com erro maior ser aceito (Figura 4). Sendo assim, quando T→0 o algoritmo se comporta como um *gradiente descendente* tradicional (detalhado mais adiante).

[7] Distribuição de Boltzmann, utilizada para calcular a probabilidade P_i de um sistema se encontrar no estado *i* sendo que este possui energia ϵ_i; *k* é a constante de Boltzmann.

4 - A convergência é obtida quando não há mais variação na função objetiva.

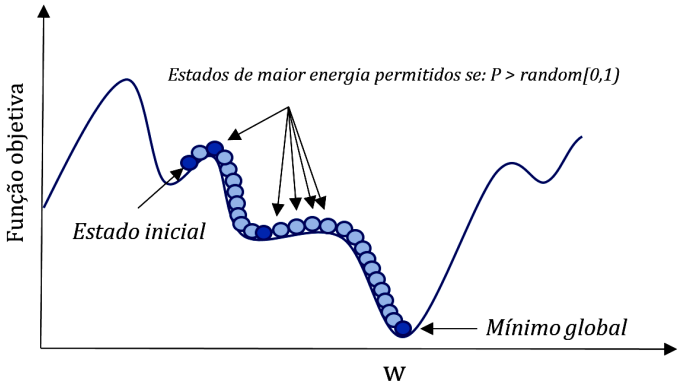


Figura 4 – Representação do processo de otimização realizado através do método Recozimento Simulado. A cada conjunto de iterações a temperatura T é abaixada, reduzindo as chances de um passo de maior valor da função objetiva ser aceito.

Backpropagation

Backpropagation ou *retropropagação*, termos geralmente utilizados como abreviação para "*backward propagation of errors",* é um algoritmo de aprendizado supervisionado desenvolvido para a otimização de Redes Neurais Artificiais (ANNs). Este algoritmo opera juntamente com o método de gradiente descendente para otimizar as ANNs. O gradiente descendente, por sua vez, é implementado a partir do gradiente da função objetiva, o qual é definido pelo conjunto das derivadas parciais da função objetiva com relação aos parâmetros em otimização. Os gradientes são utilizados para orientar a direção e tamanho dos "passos" a serem tomados pelos parâmetros. Se o objetivo é a minimização, o passo é dado em direção oposta ao gradiente (*i.e.* gradiente descendente) e se o objetivo é a maximização da função, o passo é dado na direção do gradiente (*i.e.* gradiente ascendente).

O cálculo desse gradiente é realizado partindo-se da última camada e indo em direção à primeira, ou seja, de trás para a frente, daí o termo (*back*) - *retro* propagação. Desse modo, o gradiente parcial da camada seguinte é utilizado para se computar o gradiente da camada anterior através da conhecida regra da cadeia:

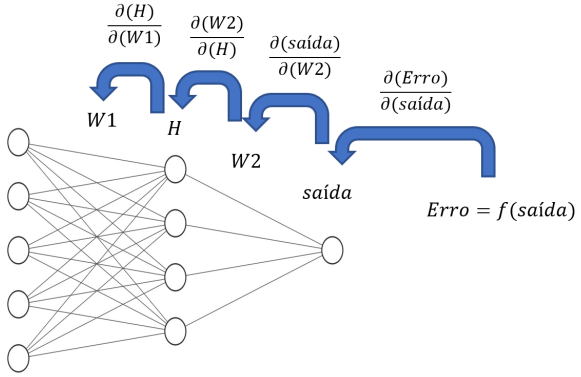


Figura 5 – Esquema de como são computados os gradientes da função erro em relação à cada uma das camadas utilizando *backpropagation*. O gradiente é usado posteriormente para a determinação dos novos valores de cada um dos pesos de maneira iterativa.

A finalização da retropropagação é alcançada quando se define o gradiente da função objetiva em relação aos parâmetros da camada mais distante dessa função, isto é, a primeira camada da rede, *W1* (Figura 5). Desse modo, definimos então para a rede acima os seguintes gradientes da função erro em relação a cada um dos conjuntos de parâmetros utilizando a *regra da cadeia das derivadas*:

$$\frac{\partial(\text{Erro})}{\partial(\text{W2})} = \frac{\partial(\text{Erro})}{\partial(\text{saída})} \cdot \frac{\partial(\text{saída})}{\partial(\text{W2})}$$

$$\frac{\partial(\text{Erro})}{\partial(\text{W1})} = \frac{\partial(\text{Erro})}{\partial(\text{saída})} \cdot \frac{\partial(\text{saída})}{\partial(\text{W2})} \cdot \frac{\partial(\text{W2})}{\partial(\text{H})} \cdot \frac{\partial(\text{H})}{\partial(\text{W1})}$$

Embora os três métodos exemplificados acima possuam o mesmo objetivo, isto é, a determinação dos valores ótimos para os parâmetros *w* através da minimização de uma *função-custo* ou *função objetiva*, geralmente o *erro médio* ou o *erro absoluto*, eles apresentam diferenças fundamentais quanto à forma com que esses valores ótimos são "procurados".

Enquanto no algoritmo *backpropagation* a busca é realizada empregando adaptações do método de gradiente descendente clássico, no Algoritmo Genético e no Recozimento Simulado são empregados métodos heurísticos [24] na busca pelo ponto ótimo no espaço de *n* dimensões constituídas pelos *n* parâmetros do modelo. Dessa forma, o *backpropagation* corresponde a um método mais eficiente na procura local pelos parâmetros ótimos, ou seja, melhor resultado na região próxima de onde os parâmetros foram inicializados, enquanto os demais métodos de otimização são desenvolvidos para apresentar melhor desempenho na busca global desses parâmetros em espaços de busca não-convexos.

De modo geral, o método *backpropagation* é considerado o mais tradicional para o treinamento das redes neurais devido a: 1) simplicidade e facilidade em programá-lo, 2) não possuir parâmetros que precisam ser pré-otimizados para o funcionamento do algoritmo, 3) ser flexível, visto que não é necessário nenhum conhecimento prévio sobre a rede a ser otimizada, 4) historicamente funcionar bem em problemas para os quais modelos baseados em ANNs são desenvolvidos, e 5) não necessitar de nenhuma informação sobre as características da função a ser otimizada.

Uma vez obtidos os gradientes da função erro em relação a cada um dos pesos *w*, o próximo passo é utilizá-los para recalcular os pesos de modo a prosseguir com a minimização da função erro computada pela rede neural. Nesse sentido, existem na literatura diversas propostas de algoritmos para a implementação do gradiente da função objetiva em métodos de minimização baseados em gradiente descendente: AdaGrad [25], RMSProp [26], SGDNesterov [27], AdaDelta [28], ADAM [29], etc. Dentre eles, o algoritmo ADAM (*Adaptive Moment Estimation* ou *Estimador de Momento Adaptativo*) tem recebido notoriedade pelo seu desempenho na área de Aprendizagem Profunda (*Deep Learning)* sobretudo em relação as demais opções de algoritmos mencionados [30].

ADAM

Na publicação original [29] foi demonstrada a eficiência do ADAM em relação aos demais algoritmos quanto as expectativas das análises teóricas. Um dos testes realizados incluiu a otimização de redes neurais do tipo processamento progressivo (*feed foward)* para a classificação de imagens de dígitos escritos à mão quanto ao valor escrito em cada uma dessas imagens. As conclusões dos autores são de que "[...] *utilizando modelos e conjuntos de dados grandes, o ADAM pode resolver problemas práticos de Aprendizagem Profunda, sendo comparativamente superior em relação a outros métodos estocásticos [9] de otimização".*

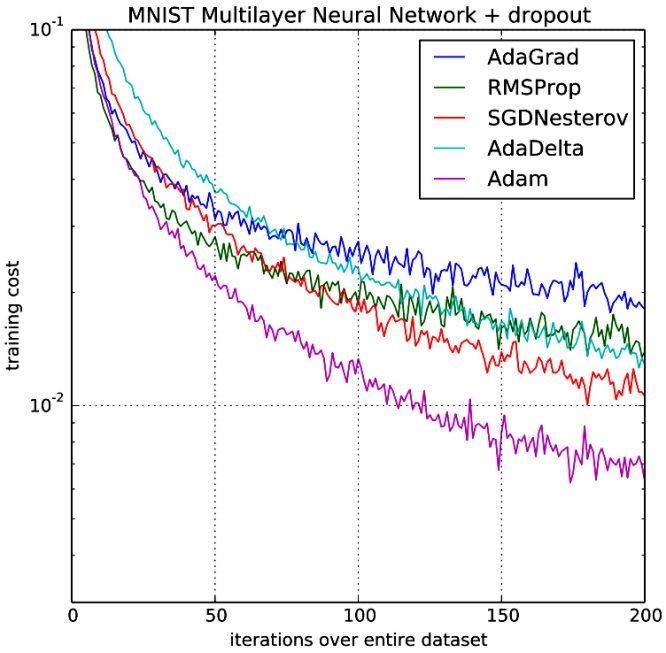


Figura 6 – Comparação do ADAM com os demais algoritmos de otimização no treinamento de uma rede neural com múltiplas camadas para a classificação de dígitos manuscritos. A imagem foi retirada do trabalho original *"Adam: A Method for Stochastic Optimization", 2015.*

O ADAM utiliza a ideia de *momento*[8] inspirado no conceito físico para definir e adaptar as taxas de aprendizado de cada um dos parâmetros ou pesos *durante* o processo de otimização. Mais precisamente, dois *momentos* são empregados para realizar essa adaptação que é, assim, feita em dois níveis diferentes:

$$(1º \text{ momento}) \qquad m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$(2º \text{ momento}) \qquad v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^{\,2}$$

Em que o subscrito t é um número inteiro e indica a iteração correspondente e g_t é o gradiente nessa iteração t da *função objetiva* em relação aos parâmetros a serem otimizados. Os termos β_1 e β_2 são os hiperparâmetros do algoritmo e controlam como os momentos se movem ao longo do tempo. Os valores mais comuns para esses dois hiperparâmetros, por apresentarem melhores resultados historicamente, são 0,9 e 0,99 respectivamente. Esses valores são praticamente fixos e muito raramente são alterados. Os termos m_t e v_t correspondentes aos momentos são também conhecidos como *médias em movimento*, e são utilizadas para atualizar o vetor de parâmetros θ do modelo como a seguir:

$$\theta_t = \theta_{t-1} - \alpha \left(\frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon} \right)$$

Em que α corresponde a taxa de aprendizado, $\epsilon = 10^{-8}$ é considerado ótimo para balancear a mudança na taxa de aprendizado α.

Os termos $\widehat{m}_t$ e $\widehat{v}_t$

são as médias em movimento descritas anteriormente corrigidas para o viés ou *bias* conforme a seguir:

$$\widehat{m}_t = \frac{m_t}{1 - \beta_1} + \frac{(1 - \beta_1) g_t}{1 - \beta_1^{\,t}}$$

$$\widehat{v}_t = \frac{v_t}{1 - \beta_2^{\,2}}$$

Finalmente, as iterações são continuadas com a recorrente atualização dos pesos (equação utilizada para atualizar o vetor de parâmetros θ, descrita acima) até a função objetiva alcançar um mínimo após *t* iterações.

De volta à solução não-ideal acetona-água

Pode-se agora verificar a aplicabilidade de um modelo baseado em uma rede neural no problema de regressão introduzido no início deste texto, isto é, o ajuste da fração em mol na fase de vapor da acetona y_1 utilizando como dados de entrada a sua fração em mol na fase líquida x_1. Pode-se então comparar os resultados obtidos com os modelos polinomial e logarítmicos já experimentados e mostrados na Figura 1. Nesse caso será utilizado o método *backpropagation* como método de otimização ou treinamento da rede, enquanto o ADAM será utilizado para processar os gradientes obtidos para recalcular os pesos a cada iteração. A rede proposta (Figura 7) foi desenvolvida manualmente (*hard coded*) utilizando a linguagem de programação PYTHON; o código para o modelo está apresentado no final deste texto. A rede possui duas camadas internas contendo 4 e 8 neurônios respectivamente. Em ambas as camadas internas foi empregada a função de ativação tangente hiperbólica. Também foram adicionados termos de soma conhecidos como *bias* (*B1*, *B2* e *B3*) em cada uma das camadas para introduzir uma flexibilização adicional à otimização do modelo.

$$output = W3.\tanh\left(\tanh(W1.x_1 + B1).W2 + B2\right) + B3$$

Os pesos do modelo acima foram inicializados com valores aleatórios no intervalo [-1,1], enquanto os hiperparâmetros para o ADAM foram inicializados como mencionado anteriormente: α = 0,001, ε = 1.10^{-8}, β_1 = 0,9 e β_2 = 0,999. Após um total de 50.000 iterações ou "*epochs*"

[8] O conceito físico de momento *p* também pode ser definido como "massa em movimento". É dado pelo produto da massa pela velocidade: *p = mv*

foi possível obter um erro médio absoluto (EMA) de 0,0011, valor muito inferior ao erro apresentado pelos modelos experimentados anteriormente (Figura 8).

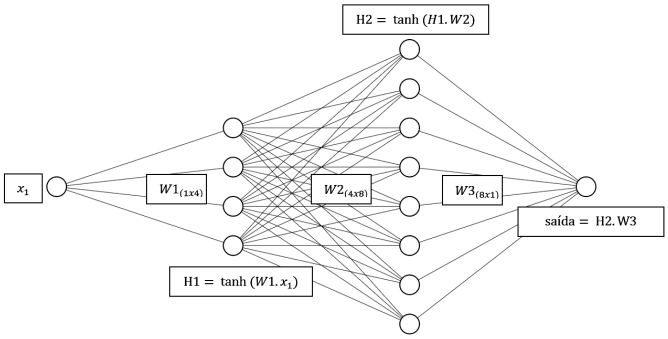


Figura 7 – Modelo de rede neural empregado para a previsão da pressão de vapor da acetona na mistura acetona-álcool. No modelo acima x_1 é a fração molar de acetona na fase líquida, W_1, W_2, W_3 são as matrizes de pesos e B_1, B_2, B_3 são os termos somados (*biases)* somados às suas respectivas camadas para flexibilizar o modelo e *saída* é o valor previsto para a fração molar da acetona na fase de vapor y_1.

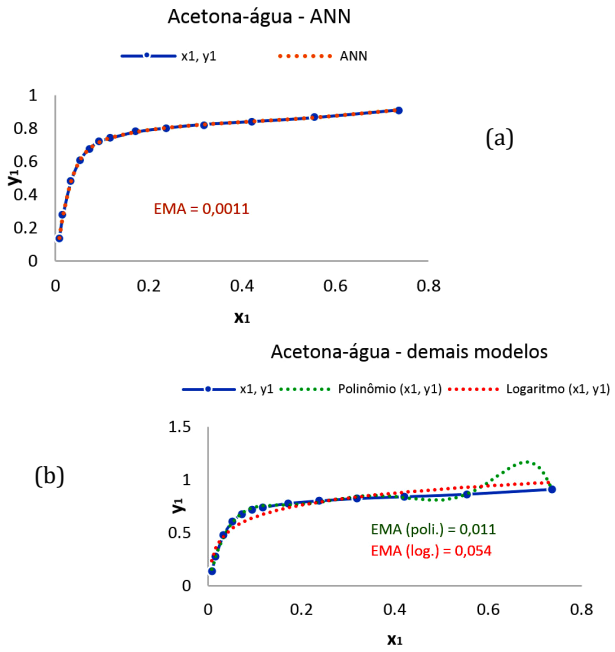


Figura 8 – Comparação do ajuste do modelo ANN (a) com os ajustes dos demais modelos (b) apresentados na Figura 1. Erro médio absoluto de 0,0011, 0,011 e 0,054 respectivamente.

$$= \begin{bmatrix} 0{,}47 \\ 0{,}20 \\ -1{,}00 \\ 0{,}43 \\ -0{,}70 \\ 1.10^{-3} \\ -0{,}60 \\ -0{,}70 \end{bmatrix}^{T} . \tanh\left\{ \tanh\left(\begin{bmatrix} -0{,}26 \\ 0{,}28 \\ 2{,}49 \\ 2{,}62 \end{bmatrix} . x_1 + \begin{bmatrix} -0{,}33 \\ 0{,}34 \\ -0{,}09 \\ 0{,}13 \end{bmatrix} \right)^{T} . \begin{bmatrix} -0{,}45 & 4.10^{-3} & -0{,}06 & -1{,}22 & 1{,}17 & -0{,}34 & 0{,}028 & 0{,}68 \\ -0{,}22 & 0{,}89 & 0{,}44 & 0{,}57 & -0{,}06 & -0{,}45 & -0{,}47 & -0{,}83 \\ -0{,}09 & 0{,}20 & -4{,}53 & 0{,}25 & 0{,}47 & -0{,}50 & 0{,}68 & 0{,}56 \\ 0{,}24 & -0{,}31 & -4{,}72 & 0{,}77 & -0{,}88 & 0{,}63 & -0{,}30 & -0{,}35 \end{bmatrix} + \begin{bmatrix} -0{,}56 \\ -0{,}09 \\ -0{,}29 \\ -0{,}59 \\ -0{,}97 \\ -0{,}10 \\ 0{,}70 \\ 0{,}47 \end{bmatrix} \right\} - 0{,}68$$

Conclusão

Embora os modelos matemáticos clássicos tenham dominado o campo da química durante muito tempo no tocante à solução de problemas de regressão, o surgimento de novas ferramentas de modelagem, sobretudo no campo do *Deep Learning*, tem preenchido lacunas deixadas por estes modelos tradicionais, ao mesmo tempo em que vêm viabilizando a modelagem de problemas muito mais complexos e atuais dentro da química, tais como: design de fármacos [31-33], descoberta de novos compostos [34-36], dinâmica molecular [37,38] e química quântica [39-41] são algumas dás áreas que mais têm explorado a aplicabilidade do *machine learning* na solução de problemas de difícil acesso por meio de métodos clássicos.

Devido a sua alta capacidade de representar funções dos mais variados tipos, as redes neurais vêm ganhando cada vez mais espaço dentro do escopo do *Deep Learning* na solução de problemas que envolvem o ajuste de funções de difícil regressão e/ou a modelagem de funções analiticamente desconhecidas. É importante mencionar, no entanto, que o poder de generalidade das redes neurais advém da complexidade matemática introduzida por este modelo, que escala rapidamente com a adição de mais *nodes* (neurônios) e mais camadas internas. Em outras palavras, aumentando-se a complexidade do modelo obtém-se uma maior exatidão, mas, ao mesmo tempo, se torna cada vez mais difícil fazer suposições a respeito das características da função que está sendo aproximada. Por essa razão, modelos de redes neurais são frequentemente associados à ideia de "caixa-preta", isto é, embora seja possível construir manualmente uma rede neural, torna-se muito difícil analisar o modelo gerado frente a grande quantidade de pesos ou parâmetros empregados. A implicação prática é que o modelo pode realizar a previsão 'correta', mas é muito difícil entender como ele o faz.

Apesar da dificuldade de se acessar o modelo se configurar em uma limitação das redes neurais, ela não deve ser confundida com a capacidade deste modelo de ser expresso matematicamente, independentemente do seu nível de complexidade. De fato, todos os modelos gerados podem ser representados em uma única equação escrita em termos dos pesos *w* empregados e conhecidos durante todo o processo de treinamento ou otimização. A equação abaixo, por exemplo, aproxima [9] através da rede neural criada anteriormente a função analítica que relaciona x_1 e y_1.

ou

$$
\begin{aligned}
y_1 = {} & 0{,}47(\tanh(-0{,}45.\, tanh(-0{,}26.\, x_1 - 0{,}33) - 0{,}22.\, tanh(0{,}28.\, x_1 + 0{,}34) - 0{,}09.\, tanh(2{,}49.\, x_1 - \\
& 0{,}09) + 0{,}24.\, tanh(2{,}62.\, x_1 + 0{,}13) - 0{,}56)) \\
& +0{,}20(\tanh\,(4.10^{-3}.\tanh(-0{,}26.\, x_1 - 0{,}33) \\
& \qquad + 0{,}89.\tanh(0{,}28.\, x_1 + 0{,}34) \\
& \qquad + 0{,}20.\tanh(2{,}49.\, x_1 - 0{,}09) - 0{,}31.\tanh(2{,}62.\, x_1 + 0{,}13) - 0{,}09)) \\
& -1{,}00(\tanh(-0{,}06.\tanh(-0{,}26.\, x_1 - 0{,}33) \\
& \qquad + 0{,}44.\tanh(0{,}28.\, x_1 + 0{,}34) \\
& \qquad - 4{,}53.\tanh(2{,}49.\, x_1 - 0{,}09) - 4{,}72.\tanh(2{,}62.\, x_1 + 0{,}13) - 0{,}29))) \\
& +0{,}43(\tanh(-1{,}22.\tanh(-0{,}26.\, x_1 - 0{,}33) \\
& \qquad + 0{,}57.\tanh(0{,}28.\, x_1 + 0{,}34) \\
& \qquad + 0{,}25.\tanh(2{,}49.\, x_1 - 0{,}09) + 0{,}77.\tanh(2{,}62.\, x_1 + 0{,}13) - 0{,}59))) \\
& -0{,}70(\tanh(1{,}17.\tanh(-0{,}26.\, x_1 - 0{,}33) \\
& \qquad - 0{,}66.\tanh(0{,}28.\, x_1 + 0{,}34) \\
& \qquad + 0{,}47.\tanh(2{,}49.\, x_1 - 0{,}09) - 0{,}88.\tanh(2{,}62.\, x_1 + 0{,}13) - 0{,}97))) \\
& +1.10^{-3}(\tanh(-0{,}34.\tanh(-0{,}26.\, x_1 - 0{,}33) \\
& \qquad - 0{,}45.\tanh(0{,}28.\, x_1 + 0{,}34) \\
& \qquad - 0{,}50.\tanh(2{,}49.\, x_1 - 0{,}09) + 0{,}63.\tanh(2{,}62.\, x_1 + 0{,}13) - 0{,}10))) \\
& -0{,}60(\tanh(0{,}028.\tanh(-0{,}26.\, x_1 - 0{,}33) \\
& \qquad - 0{,}47.\tanh(0{,}28.\, x_1 + 0{,}34) \\
& \qquad + 0{,}68.\tanh(2{,}49.\, x_1 - 0{,}09) - 0{,}30.\tanh(2{,}62.\, x_1 + 0{,}13) + 0{,}70))) \\
& -0{,}70(\tanh(0{,}68.\tanh(-0{,}26.\, x_1 - 0{,}33) \\
& \qquad - 0{,}83.\tanh(0{,}28.\, x_1 + 0{,}34) \\
& \qquad + 0{,}56.\tanh(2{,}49.\, x_1 - 0{,}09) - 0{,}35.\tanh(2{,}62.\, x_1 + 0{,}13) + 0{,}47)) - 0{,}68
\end{aligned}
$$

Dessa forma, esse tipo de "equação-modelo" além de permitir a realização das previsões para as quais o modelo foi treinado, permite ainda que ela seja utilizada como uma função substituta ou intermediária no desenvolvimento de métodos e teorias que de alguma maneira dependem de funções cujas representações analíticas são desconhecidas.

Referências

[1] - **Custodio R, Andrade JC de, Augusto F.** Curve fitting of mathematical functions to experimental data. Química Nova 1997, 20(2): 219-225.

[2] – **Reinders W, Minjet C.** Vapour equilibria in ternary systems. VI. The system water-acetone-chloroform. Recueil des Travaux Chimiques des Pays-Bas 1947, 66(9): 576-604.

[3] - **Funahashi K.** On the approximate realization of continuous mappings by neural networks. Neural Networks 1989, 2(3): 183-192.

[4] - **Lewicki G, Marino G.** Approximation by superpositions of a sigmoidal function. Zeitschrift fur Analysis und ihre Anwendung 2003, 22(2): 463-470.

[5] – **Stinchcombe, WH.** Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. Trabalho apresentado no International 1989 Joint Conference on Neural Networks; 1989 613-617 [acesso em 21 jul. 2021], Washington. Disponível em:https://ieeexplore.ieee.org/document/118640/citations?tabFilter=patents

[6] – **Cotter N.** The Stone-Weierstrass Theorem and Its Application to Neural Networks. IEEE Transactions on Neural Networks 1990, 1(4), 1990: 290-295.

[7] – **Ito Y.** Representation of functions by superpositions of a step or sigmoid function and their applications to neural network theory. Neural Networs 1991, 4(3): 385-394.

[8] – **Hornik K.** Approximation capabilities of multilayer feedforward networks. Neural Networks 1991, 4(2): 251-257.

[9] – **Hornik K, Stinchcombe, WH.** Multilayer feedforward networks are universal approximators. Neural Networks 1989, 2(5): 359-366.

[10] – **Kreinovich V.** Arbitrary nonlinearity is sufficient to represent all functions by neural networks: A theorem. Neural Networks 1991, 4(3): 381-383.

[11] – **Ripley B.** Pattern recognition and neural networks. Cambridge: Cambridge University Press, 1996.

[12] – **Silva I, Spatti D, Flauzino R.** Redes Neurais Artificiais Para Engenharia e Ciências Aplicadas. São Paulo: Artliber, 2010.

[13] – **Hassoun M.** Fundamentals of Artificial Neural Networks. Massachusetts: MIT-press, 1995.

[14] – **Sammut C, Webb G.** Encyclopedia of Machine Learning. Boston: Springer, 2010.

[15] – **Yang X.** Nature-Inspired Optimization Algorithms. 2. ed. London: Academic Press, 2011.

[16] – **Haeser G, Gomes M.** Aspectos Teóricos de Simulated Annealing e um Algoritmo duas Fases em Otimização Global. Trends in Computational and applied Mathematics 2008, 9(3): 395-404.

[17] – **Sloss A, Gustafson S.** 2019 Evolutionary Algorithms Review. Neural and Evolutionary Computing. [Internet], 2019 [acesso em 21 jul. 2021]. Disponível em: https://arxiv.org/pdf/1906.08870.pdf

[18] – **Radcliffe N.** Equivalence Class Analysis of Genetic Algorithms. Complex Systems 1991, 5(2):183-205.

[19] – **Wright A.** Genetic Algorithms for Real Parameter Optimization. Foundations of Genetic Algorithms 1991, 1: 205-218.

[20] – **Michalewics Z.** Genetic Algorithms + Data Structures = Evolution Programs. 3.ed. Berlin: Springer-Berlin-Heidelberg, 1992.

[21] – **Mühlenbein H, Schlierkamp-Voosen D.** Predictive Models for the Breeder Genetic Algorithm I. Continuous Parameter Optimization. Evolutionary Computation 1993, 1: 25-49.

[22] – **Eshelman L, Schaffer D.** Real-Coded Genetic Algorithms and Interval Schemata. Foundations of Genetic Algorithms 1993, 2: 187-202.

[23] – **Rodrigues M, Machado C, Lima M.** Simulated annealing aplicado ao problema de alocação de berços. Journal of Transport Literature 2013, 7(3): 117-136.

[24] – **Chen Y, Roux B.** Generalized Metropolis acceptance criterion for hybrid non-equilibrium molecular dynamics – Monte Carlo simulations. J. Chem. Phys. 2015, 142: 024101.

[25] – **Duchi J, Singer Y.** Adaptative Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research 2011, 12: 2121-2159.

[26] – **Graves A.** Generating Sequences with Recurrent Neural Networks. Neural and Evolutionary Computing. [Internet], 2013 [acesso em 21 jul 2021]. Disponível em: https://arxiv.org/abs/1308.0850

[27] – **Yu, E.** A method of solving a convex programming problem with convergence rate O(1/k^2). Dokl. Akad. Nauk SSSR 1983, 269(3): 543-547.

[28] – **Zeiler M.** ADADELTA: An adaptive Learning Rate Method. Machine Learning. [Internet], 2012 [acesso em 21 jul 2021]. Disponível em: https://arxiv.org/abs/1212.5701

[29] – **Kingma D, Ba J.** Adam: A method for stochastic optimization. [Internet], 2017 [acesso em 21 jul 2021]. Disponível em: https://arxiv.org/abs/1412.6980

[30] – **Ruder S.** An overview of gradient descente optimization algorithms. [Internet]. 2016 [acesso em 21 jul 2021], Disponível em: https://arxiv.org/abs/1609.04747

[31] – **Vamathevan J, Clark D, Czodrowski P, Dunham I, Edgardo F, George L. et al.** Applications of machine learning in drug discovery and development. Nature Reviews Drug Discovery 2019, 18(6): 463-477.

[32] – **Patel L, Shukla T, Huang X, Ussery D, Wang S.** Machine Learning Methods in Drug Discovery. Molecules 2020, 25(22): 5277.

[33] – **Chen, H, Engkvist O, Wang Y, Olivecrona M, Blaschke T.** The rise of deep learning in drug discovery. Drug Discovery Today 2018, 23(6): 1241-1250.

[34] – **Tkatchenko A.** Machine learning for chemical discovery. Nature Communications 2020, 11: 1-4.

[35] – **von Lilienfeld O, Burke K.** Retrospective on a decade of machine learning for chemical discovery. Nature Communications 2020, 11: 1-4.

[36] – **Cova T, Pais A.** Deep Learning for Deep Chemistry: Optimizing the Prediction of Chemical Patterns. Frontiers in Chemistry 2019, 7: 809.

[37] – **Gastegger M, Behler P, Marquetand.** Machine learning molecular dynamics for the simulation of infrared spectra. Chemical Science 2017, 8(10): 6924-6935.

[38] – **Gebhardt J, Kiesel M, Riniker S, Hansen N.** Combining Molecular Dynamics and Machine Learning to Predict Self-Solvation Free Energies and Limiting Activity Coefficients. Journal of Chemical Information and Modeling 2020, 60(11): 5319-5330.

[39] – **Li H, Collins C, Tanha M, Gordon J, Yaron D.** A Density Functional Tight Binding Layer for Deep Learning of Chemical Hamiltonians. Journal of Chemical Theory and Computation 2018, 14(11): 5764-5776.

[40] – **von Lilienfeld O, Ramakrishnan R, Rupp M, Knoll A.** Fourier series of atomic radial distribution functions: A molecular fingerprint for machine learning models of quantum chemical properties. International Journal of Quantum Chemistry 2015, 115(16): 1084-1093.

[41] – **Lopez-Bezanilla A, von Lilienfeld O.** Modeling electronic quantum transport with machine learning. Physical Review B – Condensed Matter and Materials Physics 2014, 89: 235411.

Apêndice

Código Python (versão 3.8) para a criação e treinamento do modelo de rede neural referente ao sistema acetona-água apresentado

Para fins de simplificação as derivadas para o cálculo do gradiente foram computadas de maneira numérica utilizando o método da derivada simétrica

```
# numpy é usado para converter as listas numéricas em vetores e matrizes
import numpy as np
# njit é uma ferramenta utilizada para acelerar as iterações e reduzir o tempo gasto
from numba import njit
# o módulo random é usado para inicializar os pesos da rede
import random

# listas contendo as frações molares na fase líquida e de vapor respectivamente
X = [0.008,0.016,0.033,0.052,0.072,0.094,0.117,0.171,0.237,0.318,0.42,0.554,0.736]
Y = [0.138,0.277,0.479,0.604,0.675,0.719,0.738,0.776,0.8,0.822,0.839,0.863,0.909]
X = np.array(X)
Y = np.array(Y)

# lista com a inicialização dos 57 parâmetros ou pesos da rede
p = [random.uniform(-1,1) for r in range(0, 57)]

# hiperparâmetros inicializados para o Adam
m = np.zeros(57)
v = np.zeros(57)
t = 0
alpha = 0.001
epsilon = 10**(-8)
beta1 = 0.9
beta2 = 0.999


@njit
# função para o cálculo do erro acumulado nas previsões
def error(B):
  sume = np.zeros_like(X)
  count = 0
# matrizes de pesos e bias (Figura 7) inicializadas
  w1, b1, w2, b2, w3, b3 = np.ones((4, 1)), np.ones(4), np.ones((4, 8)), np.ones(8),np.ones((8,1)), np.ones(1)

# preenchimento das matrizes inicializadas acima com os valores contidos no vetor de parâmetros 'p'
inicializado anteriormente

  for p in range(len(w1)):
    for q in range(len(w1[p])):
      w1[p][q] = B[count]
      count += 1
  for p in range(len(b1)):
    b1[p] = B[count]
    count += 1
  for p in range(len(w2)):
    for q in range(len(w2[p])):
      w2[p][q] = B[count]
      count += 1
  for p in range(len(b2)):
    b2[p] = B[count]
    count += 1
  for p in range(len(w3)):
    for q in range(len(w3[p])):
      w3[p][q] = B[count]
      count += 1
  for p in range(len(b3)):
    b3[p] = B[count]
    count += 1
```

```
# combinação das matrizes para a composição do modelo e cálculo do erro cumulativo

    for n in range(0, len(X)):
        camada1 = np.tanh(w1.dot(np.array([X[n]])) + b1)
        camada2 = np.tanh(np.dot(camada1, w2) + b2)
        output = np.dot(camada2, w3) + b3
        sume[n] = (((output - Y[n])**2)**0.5)[0]

    error = np.sum(sume)/len(X)
    #print(error)
    return error


# realização das iterações e atualização do vetor 'p' de parâmetros através do Adam

for iteration in range(0, 50000):
    t+=1
    print('iteração', t)
    g = []
# cálculo das derivadas numéricas para a composição do gradiente
    for i in range(0, len(p)):
        h = (epsilon)**0.5 * abs(p[i])
        H = []
        for k in range(0, len(p)):
            if k == i:
                H.append(h)
            else:
                H.append(0)
        p = np.array(p)
        H = np.array(H)
        g.append((error(p + H) - error(p - H))/(2*h))

# atualização dos hiperparâmetros do Adam
    m = beta1*m + (1 - beta1)*np.array(g)
    v = beta2*v + (1 - beta2)*np.power(g,2)
    m_hat = m/(1-beta1**t) + (1-beta1)*np.array(g)/(1-np.power(beta1,t))
    v_hat = v/(1-beta2**t)
# atualização do vetor de pesos 'p'
    p = p - alpha*(m_hat/(np.sqrt(v_hat) + epsilon))
```