Maria Inês Vasconcellos Furtado

Redes Neurais Artificiais: ***Uma Abordagem Para Sala de Aula***

Atena Editora
Ano 2019

Maria Inês Vasconcellos Furtado

Redes Neurais Artificiais: Uma Abordagem Para Sala de Aula

Atena Editora
2019

2019 *by Atena Editora*

Editora Chefe: Profª Drª Antonella Carvalho de Oliveira
Diagramação e Edição de Arte: Lorena Prestes
Revisão: Os autores

Dados Internacionais de Catalogação na Publicação (CIP)
(eDOC BRASIL, Belo Horizonte/MG)

F992r

Furtado, Maria Inês Vasconcellos.
Redes neurais artificiais [recurso eletrônico] : uma abordagem para sala de aula / Maria Inês Vasconcellos Furtado. – Ponta Grossa (PR): Atena Editora, 2019.

Formato: PDF
Requisitos de sistema: Adobe Acrobat Reader
Modo de acesso: World Wide Web
Inclui bibliografia
ISBN 978-85-7247-326-2
DOI 10.22533/at.ed.262191504

1. Genética – Métodos de simulação. 2. Redes neurais (Computação). I. Título.

CDD 001.53

Elaborado por Maurício Amormino Júnior – CRB6/2422

Aos meus meninos: Marco e Vinícius.

SUMÁRIO

INTRODUÇÃO ÀS REDES NEURAIS

A simulação cognitiva é uma técnica que permite imitar as estruturas e os mecanismos de raciocínio utilizados pelos operadores no cumprimento de suas atividades, estando inspirada em conceitos desenvolvidos na modelagem cognitiva, e utiliza os formalismos de representação de estruturas de domínio da Inteligência Artificial, que é um vasto campo que contém diversos componentes importantes, como as Redes Neurais Artificiais.

Uma Rede Neural Artificial consiste em uma estrutura conexionista, na qual o processamento é distribuído por um grande número de pequenas unidades densamente interligadas. Este paradigma procura entender e emular as propriedades decorrentes do alto grau de paralelismo e conectividade dos sistemas biológicos.

Uma rede neural é composta por um elevado número de elementos processadores, os neurônios, amplamente interligados através de conexões com um determinado valor que estabelece o grau de conectividade entre estes, denominado peso da conexão ou sinapse.

Deste modo, todo o processamento é realizado distributivamente entre os elementos processadores da rede, onde cada qual o realiza isolada e paralelamente, enviando seu resultado para outras unidades através das conexões entre eles. Embora cada neurônio faça um processamento bastante simples, a associação os capacita a realizar problemas altamente complexos.

A capacidade de resolver um determinado problema encontra-se na sua arquitetura, ou seja, no número e modo pelo qual os elementos processadores estão interconectados, nos pesos destas conexões e no número de camadas.

Para sintetizar uma rede neural, existem alguns métodos - dos quais o mais usualmente utilizado é o empírico - que geram um padrão de interconexão para resolver o problema através de algum processo de treinamento capaz de modificá-lo gradualmente, de modo a adaptá-lo à deliberação deste problema. A síntese é um fator determinante do sucesso ou fracasso do treinamento das redes neurais artificiais.

Os mecanismos de aprendizado possibilitam a modificação do padrão de interconexão. Para treinamento podem ser utilizados os mecanismos de aprendizado supervisionado, por reforço ou não-supervisionado, conforme a arquitetura neural fixada.

Portanto, faz-se necessárias 3 fases para aplicar redes neurais artificiais à resolução de um problema qualquer. O treinamento que ensina a rede a resolver um conjunto de padrões de saída associados a padrões de entrada; o teste, em que são apresentados padrões de entrada à rede, e as saídas obtidas são comparadas às saídas desejadas; e a aplicação, em que a rede sintetizada é utilizada na resolução do determinado tipo de problema. Dentre as possibilidades de utilização mais exploradas das redes neurais, estão:

→ a classificação de padrões de entradas em diferentes categorias;

→ o reconhecimento de um determinado número de padrões de entradas, reproduzindo-os nas suas saídas;

→ a predição de uma variável a partir de valores históricos desta ou de outras variáveis com ela relacionadas. Isto é comumente encontrado em áreas como a economia, particularmente engenharia financeira. Neste domínio, sistemas baseados em redes neurais se têm-se mostrado, muitas vezes, mais eficazes que técnicas tradicionais.

Muitas outras aplicações são constantemente encontradas para as Redes Neurais Artificiais, ao mesmo tempo que os seus fundamentos teóricos vão se tornando cada vez mais sólidos devido ao trabalho incessante de um número cada vez maior de pesquisadores.

As áreas de aplicação são as mais diversas, com grandes e pequenas aplicações em campos diversos, como engenharia, economia, agronomia, medicina etc., resolvendo problemas que envolvam extração de características, classificação, categorização/clusterização; estimativa e previsão; otimização; aproximação de funções; dentre outras.

HISTÓRICO DE REDES NEURAIS

As primeiras informações sobre a neuro computação datam de 1943, em artigos de McCulloch e Pitts, em que sugeriam a construção de uma máquina baseada ou inspirada no cérebro humano.

Em 1949 Donald Hebb escreveu um livro intitulado “A Organização do Comportamento”. Suas ideias não eram completamente novas, mas Hebb foi o primeiro a propor uma lei de aprendizagem específica para as sinapses dos neurônios.

Em 1951 foi construído primeiro neuro computador, denominado Snark, por Mavin Minsky. O Snark operava com sucesso, ajustando seus pesos automaticamente, entretanto, nunca executou qualquer função de processamento de informação interessante, mas serviu de inspiração para as ideias de estruturas que o sucederam.

Em 1956 nasceram os dois paradigmas da Inteligência Artificial: Simbólica e Conexionista. A Inteligência Artificial Simbólica tenta simular o comportamento inteligente humano desconsiderando os mecanismos responsáveis por tal, ou seja, não possui uma inspiração biológica. A Inteligência Artificial Conexionista acredita que construindo um sistema que simule a estrutura do cérebro, este sistema apresentará inteligência, será capaz de aprender, assimilar, errar e aprender com seus erros.

Em 1957 e 1958, surgiu o primeiro neuro computador a obter sucesso (Mark I Perceptron) criado por Frank Rosenblatt, Charles Wightman e outros. Seu interesse inicial para a criação do Perceptron era o reconhecimento de padrões.

Após Rosenblatt, Bernard Widrow, com a ajuda de alguns estudantes, desenvolveu um novo tipo de elemento de processamento de redes neurais chamado de Adaline, equipado com uma poderosa lei de aprendizado, que diferente do Perceptron ainda permanece em uso. Widrow fundou a primeira companhia de hardware de neuro computadores e componentes.

Os anos seguintes foram marcados por um entusiasmo exagerado de muitos pesquisadores, que prometiam máquinas tão poderosas quanto o cérebro humano e que surgiriam em um curto espaço de tempo. Isto tirou a credibilidade dos estudos desta área e causou grandes aborrecimentos aos técnicos de outras áreas.

Um período de pesquisa silenciosa seguiu-se entre 1967 e 1982, com poucas publicações devido aos fatos ocorridos anteriormente. Entretanto, aqueles que pesquisavam nesta época, e todos os que se seguiram no decorrer destes anos conseguiram novamente estabelecer um campo concreto para o renascimento da área.

Skurnick, um administrador de programas da DARPA (Defense Advanced Research Projects Agency), fundou em 1983 um centro de pesquisas em neuro computação. Este ato não só abriu as portas para a neuro computação, como também deu à DARPA o status de uma das líderes mundiais em se tratando de tecnologia.

John Hopfield, renomado físico de reputação mundial, se interessou pela neuro computação, e incentivou e difundiu os princípios desta área entre importantes cientistas, matemáticos e tecnólogos altamente qualificados.

Em 1986 o campo de pesquisa se extasiou com a publicação do livro "Processamento Distribuído Paralelo" editado por David Rumelhart e James McClelland.

Em 1987 ocorreu em São Francisco a primeira conferência de redes neurais, a IEEE International Conference on Neural Networks, e, além disso, foi formada a International Neural Networks Society (INNS).

Desde 1987, muitas universidades anunciaram a formação de institutos de pesquisa e programas de educação em neuro computação.

INSPIRAÇÃO BIOLÓGICA

Durante a evolução dos seres vivos, a multicelularidade levou a especialização de funções aos diversos tecidos orgânicos. Concomitantemente, apareceram sistemas para coordenar as atividades dos diversos tecidos e órgãos. O sistema nervoso é um dos responsáveis por esta coordenação.

O tecido nervoso é constituído por dois componentes principais: os neurônios e as células da glia ou neuroglia (Figura 1). A neuroglia compõe-se de diversos tipos celulares, que ocupam espaços entre os neurônios, atuando na sustentação, nutrição e defesa destes. Além disto, as células da neuroglia são essenciais na formação de circuitos neurais: na vida embrionária, estas células participam da orientação do crescimento dos dendritos e axônios, levando ao estabelecimento de sinapses adequadas do ponto de vista funcional; no adulto, as células da glia envolvem completamente os neurônios, indicando um papel isolante, o que possibilita a formação de circuitos independentes, impedindo a propagação desordenada dos impulsos que os percorrem.

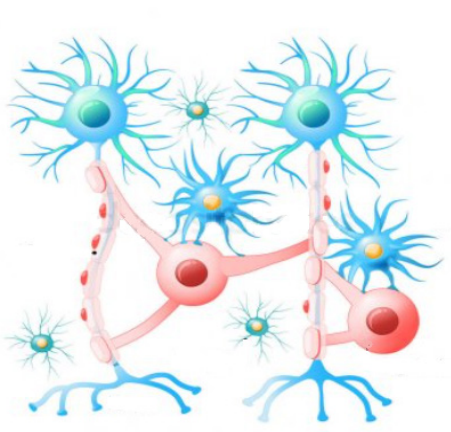


Figura 1 - Desenho esquemático de células da neuroglia.

Os neurônios ou células nervosas são a unidade básica da neuroanatomia. Apresentam a propriedade de responder a alterações do meio em que se encontram (estímulos), através de modificações da diferença de potencial elétrico que existe entre as superfícies interna e externa da membrana celular. Eles reagem prontamente a estes estímulos, e a mudança de potencial propaga-se a outros neurônios, músculos ou glândulas (impulso nervoso). São formados por um corpo celular ou soma, que contém o núcleo, constituindo o centro metabólico do neurônio. Funcionam também como um receptor e integrador de estímulos. Do corpo partem prolongamentos numerosos, os dendritos (do grego, *dentron* = árvore), especializados na função de receber estímulos (figura 2). Eles aumentam consideravelmente a superfície celular, tornando possível receber e integrar impulsos trazidos por numerosos terminais axônicos. O axônio (do grego, *áxon* = eixo) é um prolongamento único, especializado na condução de impulsos que transmitem informações a outras

células nervosas, musculares ou glandulares. Assim, em geral, as informações são recebidas pelos dendritos e corpo celular e emitidas pelo axônio, apresentados nas figuras 3a e 3b.

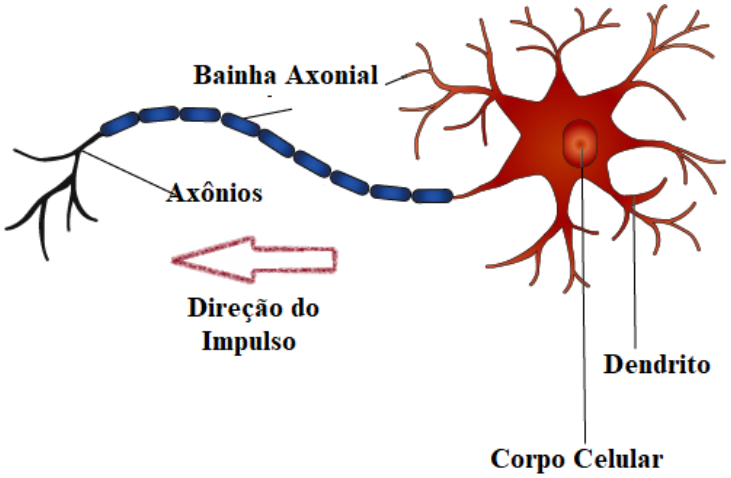
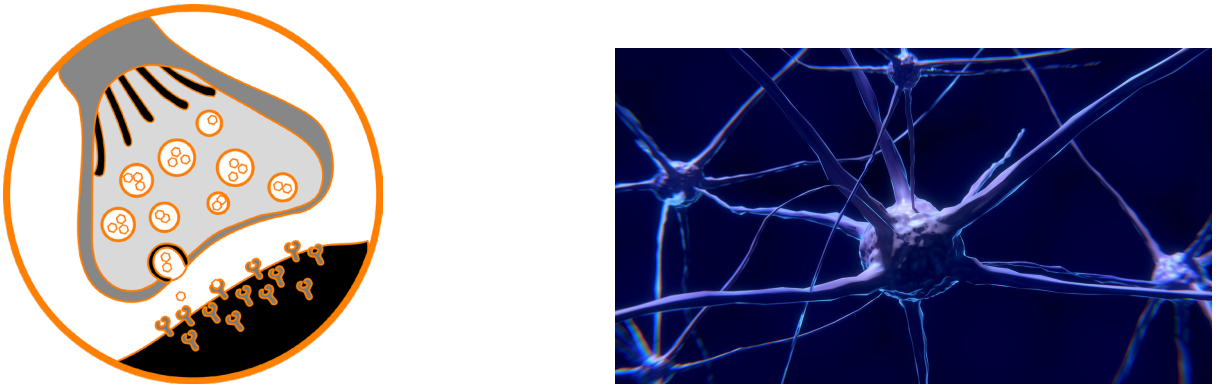


Figura 2 – Neurônio esquematizado

Durante a evolução ocorreu um aumento no número e na complexidade dos interneurônios, bem como do sistema nervoso e permitindo a realização de padrões de comportamento cada vez mais elaborados. Eles são de grande importância nos processos de aprendizado.

O número de neurônios e sua relativa abundância em diferentes partes do cérebro é um determinante da função neural e, consequentemente, do comportamento. Assim, filos cujos membros têm cérebros maiores e mais neurônios respondem às alterações ambientais com maior variação e versatilidade de comportamentos.

A transmissão dinâmica do impulso nervoso de um neurônio para outro depende de estruturas altamente especializadas, as sinapses. As sinapses são locais de contato de um axônio com um dendrito ou corpo celular de outro neurônio, podendo ser excitatórias ou inibitórias. Nelas, as membranas das duas células (ditas membranas pré e pós-sinápticas) ficam separadas por espaços de 20 a 30 nm, a fenda sináptica. Na porção terminal do axônio, observa-se numerosas vesículas que contêm neurotransmissores, que são substâncias químicas responsáveis pela transmissão do impulso nervoso através das sinapses. Estes neurotransmissores são liberados na fenda sináptica e se aderem a moléculas receptoras na membrana pós-sináptica, promovendo a condução do impulso no intervalo sináptico. A figura (3a) e (3b) apresentam sinapses.



(a) (b)

Figura 3 - (a) e (b) Sinapses.

O sistema nervoso tem dois componentes: o Sistema Nervoso Central (SNC), composto pelo encéfalo e pela medula espinhal, e o Sistema Nervoso Periférico (SNP), pelos nervos, gânglios e terminações nervosas. A figura 4 apresenta um esquema para a divisão anatômica do sistema nervoso.

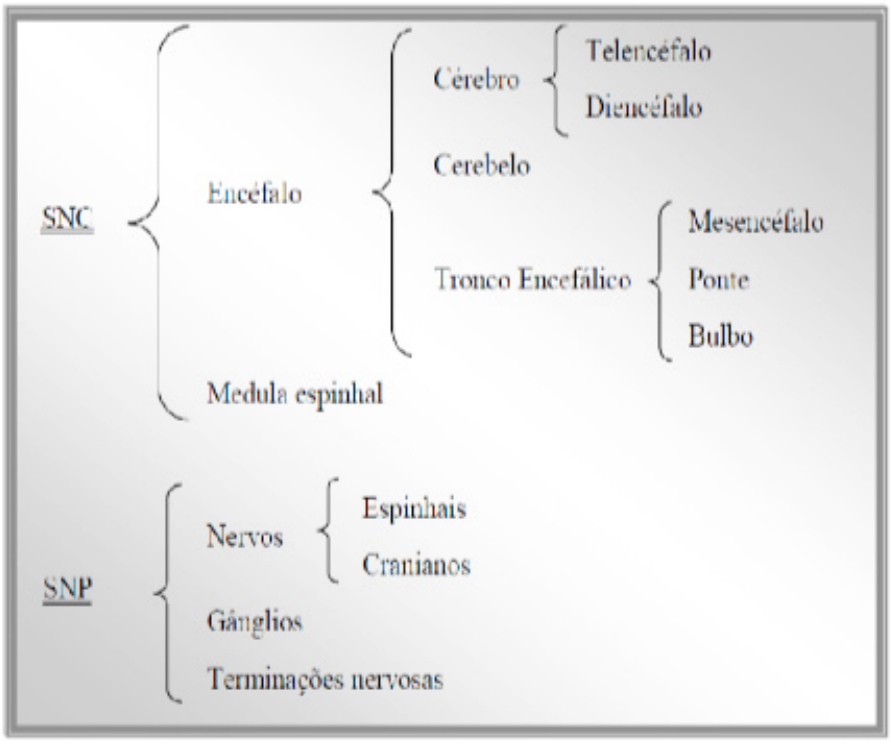


Figura 4 - Divisão anatômica do sistema nervoso

Anatomicamente estes dois componentes estão separados, mas funcionalmente são interconectados. O sistema nervoso periférico retransmite informações para o sistema nervoso central e executa comandos gerados neste. A ação mais simples pressupõe a atividade integrada de várias vias. As regiões do cérebro são especializadas para diferentes funções. Funções complexas são possíveis pelas conexões em série e paralelo de diversas regiões cerebrais (Figura 5). O cérebro humano possui cerca de

10^{11} de neurônios e cada neurônio pode receber de 1.000 a 10.000 contatos sinápticos em seu corpo e dendritos. O aprendizado pode determinar alterações estruturais no cérebro, provavelmente por causar alteração no padrão de interconexões dos diversos sistemas sensoriais e motores. Assim, nos humanos adultos, os mapas corticais não são estáticos, mas dinâmicos. As alterações produzidas pelo aprendizado e, portanto, pela formação de novas sinapses podem contribuir para a expressão biológica da individualidade.

Figura 5 – Áreas distintas do cérebro humano.

Sob a ótica computacional, pode-se dizer que em um neurônio é realizado o processamento sobre uma ou, geralmente, várias entradas, afim de gerar uma saída.

O neurônio artificial é uma estrutura lógico-matemática que procura simular a forma, o comportamento e as funções de um neurônio biológico. Pode-se, a grosso modo, associar o dendrito à entrada, o soma ao processamento e o axônio à saída; portanto, o neurônio é considerado uma unidade fundamental processadora de informação.

Os dendritos são as entradas, cujas ligações com o corpo celular artificial são realizadas através de canais de comunicação que estão associados a um determinado peso (simulando as sinapses). Os estímulos captados pelas entradas são processados pela função do soma, e o limiar de disparo do neurônio biológico é substituído pela função de transferência. A figura 6 apresenta esquematicamente um neurônio de McCullock e Pitts.

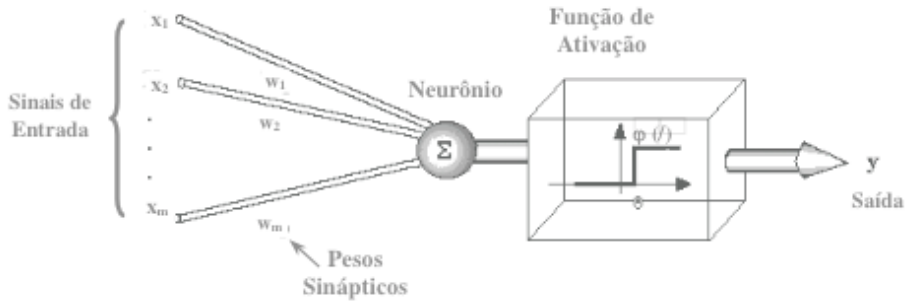


Figura 6 – Esquema de Unidade Processadora de McCullock e Pitts.

O neurônio de McCullock e Pitts, apesar de simples, ainda é utilizado. Nele os sinais são apresentados à entrada x_j; o sinal da entrada apresentado à sinapse j conectada ao neurônio k é multiplicado por um peso sináptico w_{kj}; no neurônio k, é feita a soma ponderada dos sinais recebidos, produzindo um determinado nível de atividade. Se este nível de atividade exceder um certo limite, a unidade processadora produzirá uma determinada resposta como saída.

Matematicamente o modelo apresentado na figura 6 pode ser representado por:

$$u_k = \sum_{j=1}^{m} w_{kj} \cdot x_j \qquad \text{(I)}$$

e

$$y_k = \varphi(u_k) \qquad \text{(II)}$$

onde: x_j – sinais de entrada

w_{kj} – pesos sinápticos ou pesos

y_k – sinais de saída

φ (0) – função de ativação

Para uma maior fidelidade ao modelo biológico e flexibilidade computacional, além da excitação vinda das entradas da rede ou saídas de outros neurônios, cada neurônio é também excitado por uma polarização constante chamada "*bias*", b_k constante de valor 1, transmitida ao neurônio através da sinapse $w_{i,0}$. A equação matemática (II) apresentada ficaria como na equação (III).

$$y_k = \varphi\left(u_k + b_k\right) \qquad \text{(III)}$$

Este modelo se apresenta constante para quase todas as Redes Neurais, variando somente a função de ativação. Esta limita a amplitude do sinal de saída do neurônio. Normalmente a faixa de saída está em um intervalo fechado [0, 1] ou alternativamente em [-1, 1], podendo também este intervalo de saída estar entre (- ∞, + ∞). Entre os diversos tipos de funções de ativação, as mais comuns estão representadas na figura 7.

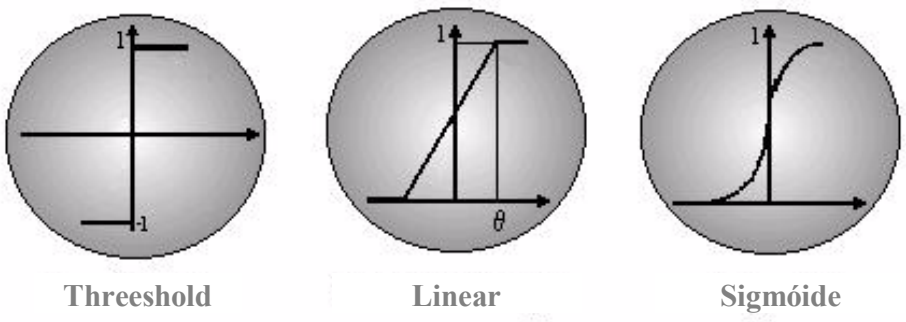


Figura 7 – Funções de Ativação.

Embora cada neurônio tenha a função de realizar um processamento simples, uma rede com múltiplos neurônios é capaz de realizar operações bastante complexas.

Uma das primeiras e mais simples formas de RNA´s foram os *perceptrons*. Em um *perceptron*, os neurônios estão dispostos em várias camadas, que podem ser classificadas em camadas de entrada, onde os padrões são apresentados à rede; camadas ocultas, escondidas ou intermediárias, onde é feita a maior parte do processamento; e a camada de saída, onde a conclusão do processamento é apresentada.

Normalmente pode-se encontrar a representação que produzirá o mapeamento correto da entrada para a saída através das unidades intermediárias. Mas foi provado que são necessárias, no máximo, duas camadas intermediárias, com um número suficiente de unidades por camada, para se produzir quaisquer mapeamentos.

As entradas da Rede Neural, simulando a captação de estímulos, podem ser conectadas em muitos neurônios, resultando em uma série de saídas, onde cada neurônio representa uma saída. Essas conexões, em comparação com o sistema biológico, representam o contato dos dendritos com outros neurônios, formando assim as sinapses. A função da conexão em si é transformar o sinal de saída de um neurônio em um sinal de entrada de outro, ou, ainda, orientar o sinal de saída para o exterior. As diferentes possibilidades de conexões entre as camadas de neurônios podem gerar diversas estruturas ou arquiteturas diferentes. A figura 8 representa um destes tipos de arquitetura, na qual é apresentada uma Rede Neural Artificial com uma camada oculta.

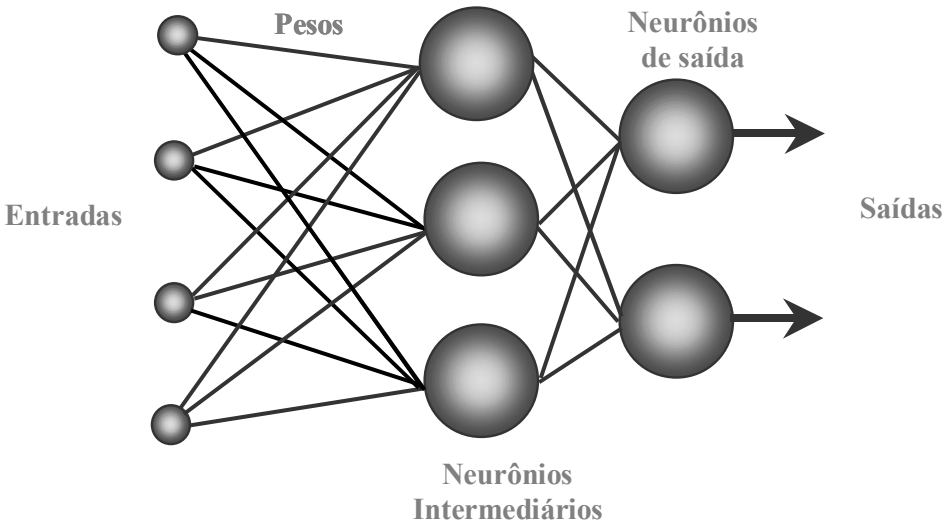


Figura 8 - Rede Neural Artificial de múltiplas camadas.

Estes modelos conexionistas modificam-se de muitas formas, conforme a necessidade de aplicação. A arquitetura pode variar através das diferentes conexões entre as camadas: pelo número de camadas intermediárias; pelo número de unidades processadoras (neurônios ou nós); pela função de transferência; e pelo processo de aprendizado. Dentre estas variantes algumas já foram comentadas.

A maneira como os neurônios de uma rede são estruturados está intimamente relacionada com o algoritmo de aprendizagem usado para treinar a rede.

A) Tipos de Arquiteturas

Antes de mostrar os algoritmos, serão apresentadas as arquiteturas fundamentais. Em geral elas são subdivididas em 3 classes: Rede Neural *Feedforward* de 1 camada, Rede Neural Feedforward Multicamadas e Redes Recorrentes ou Realimentadas.

A. Rede Feedforward de 1 camada

Neste tipo de rede os neurônios são organizados em forma de camadas, como pode ser visto na figura 9. Os nós da camada de entrada se comunicam diretamente com a camada de saída (nós computacionais).

A arquitetura do tipo feedforward em camadas apresenta uma organização similar à do córtex humano, onde os neurônios se dispõem em camadas paralelas e consecutivas, e os axônios se estendem sempre no mesmo sentido, isto é, a informação propaga-se da entrada para a saída, não existindo portanto ligações entre os neurônios de uma mesma camada ou com camadas anteriores.

É chamada de rede de 1 camada em referência à camada de saída, pois os nós da entrada não processam nada, só apresentam os padrões à rede.

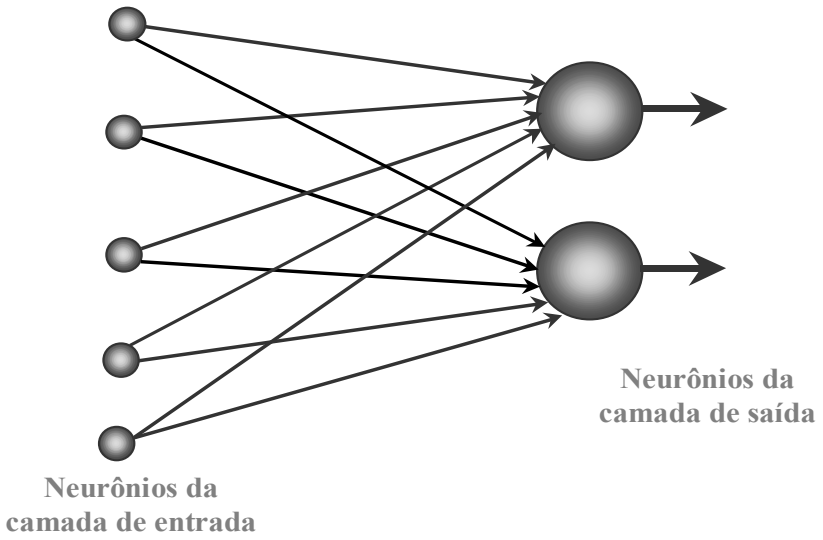


Figura 9 – Rede Feedforward de 1 camada.

B. Rede Feedforward Multicamadas

A segunda classe de redes *feedforward* distingue-se da anterior por apresentar uma ou mais camadas escondidas. A função dos neurônios escondidos é intervir entre a camada de entrada e a de saída da rede de alguma maneira útil. Pela adição de uma ou mais camadas, a rede passa a melhor mapear problemas mais complexos.

Por ser uma rede do tipo *feedforward,* as conexões se dão sempre no sentido da camada de entrada para a de saída. Quando a rede possuir todos os nós de uma camada comunicando-se com todos os nós da camada posterior, ela é dita *totalmente conectada*. Caso alguma das conexões sinápticas não esteja ligada com a camada

subsequente, a rede é dita *parcialmente conectada*. A figura 10 representa uma rede *feedforward* multicamada totalmente conectada.

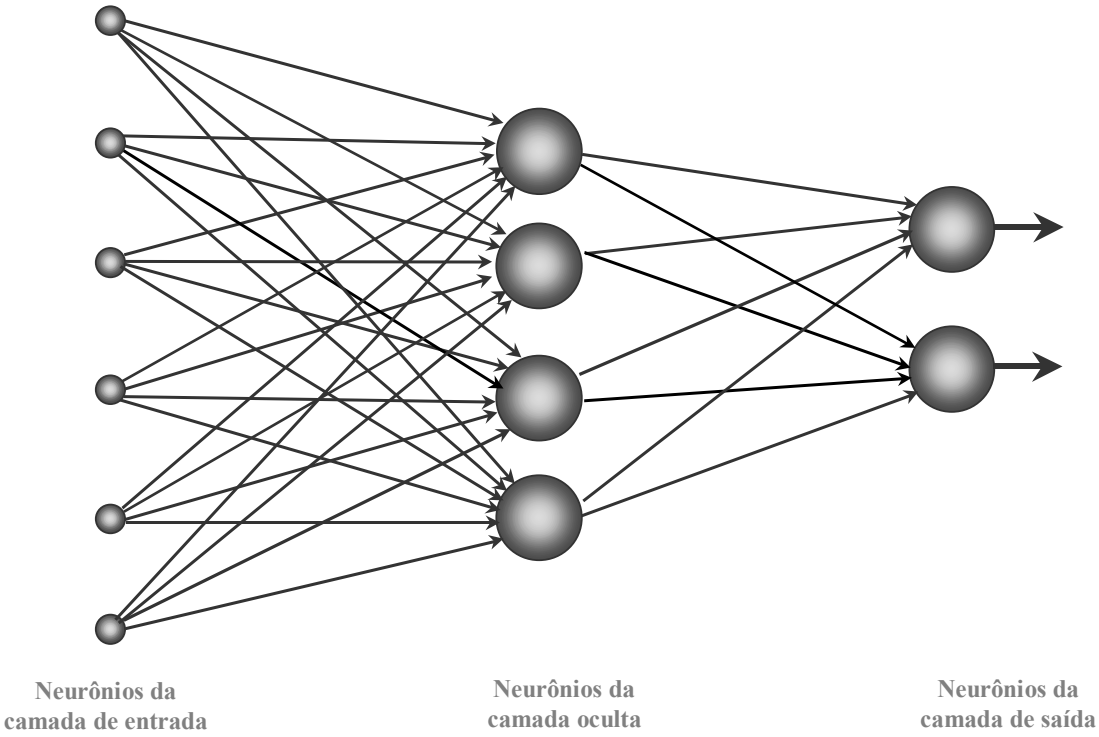


Figura 10 – Rede Feedforward Multicamadas Totalmente Conectada

C. Rede Recorrente ou Realimentada

Este tipo de rede distingue-se das redes neurais do tipo *feedforward* por permitir a realimentação de uma camada com as informações geradas pela camada posterior, ou ainda por fazer uma realimentação do neurônio com a sua própria saída (*self-feedback*).

Para permitir essa realimentação, um dispositivo de atraso é introduzido, guardando as informações de saída de um instante anterior, até que ela possa ser fornecida como entrada do instante atual. A figura 11 apresenta uma rede recorrente com *self-feedback* e sem neurônios na camada intermediária.

As redes do tipo *feedforward* podem ser consideradas um caso particular das redes realimentadas. Estas últimas são mais poderosas, porém são bem mais complexas, tanto para utilização quanto para a análise dos resultados apresentados.

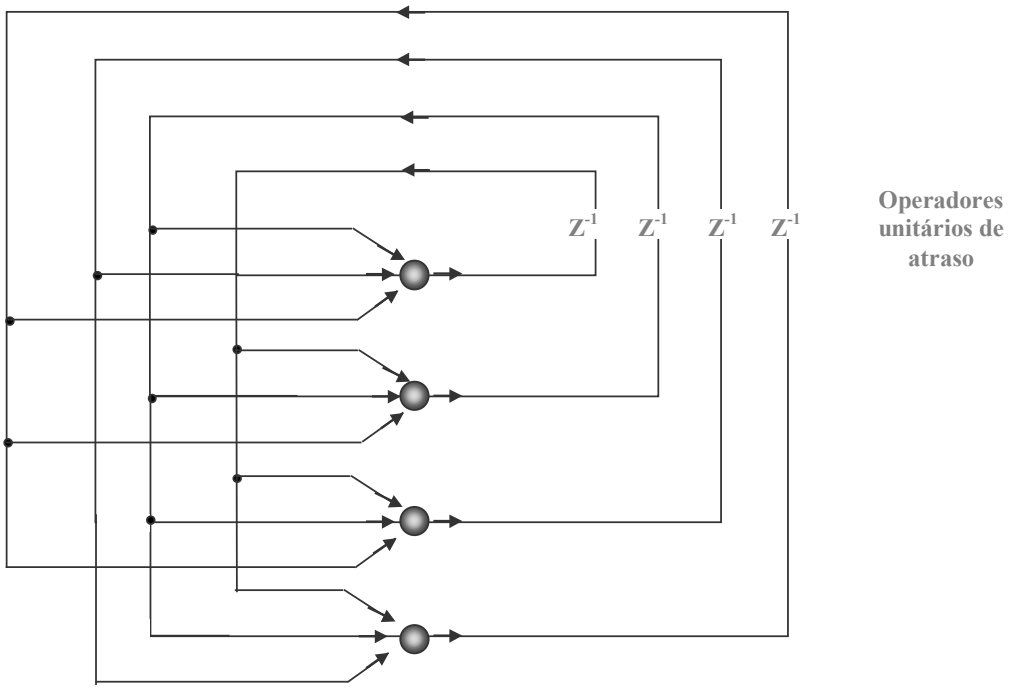


Figura 11 – Rede Recorrente.

D. Rede GRNN

Regressão Generalizada em Redes Neurais (GRNN) é um paradigma de redes desenvolvido originalmente para fins estatísticos e "redescoberto" por Donald Specht, do Laboratório de Pesquisa Lockheed de Palo Alto, em 1990. É usada principalmente para modelagem de sistemas e predição, assim como as redes *backpropagation*, Redes RBF (Função de Base Radial) e Redes de Reforço.

São do tipo *feedforward* e baseadas na avaliação da função probabilidade, treinam rapidamente e permitem a modelagem de funções não-lineares. A figura 12 apresenta esquematicamente uma rede do tipo GRNN.

GRNN é uma generalização de uma Rede Neural Probabilística (PNN) e pode ser usada em lugar desta para solucionar problemas de classificação. Porém, a PNN especificamente é destinada para resolver este tipo de problema, possuindo a habilidade de treinar sob conjunto de dados escassos, em só um passo de treinamento. PNN separa os dados em um número específico de categorias de saída.

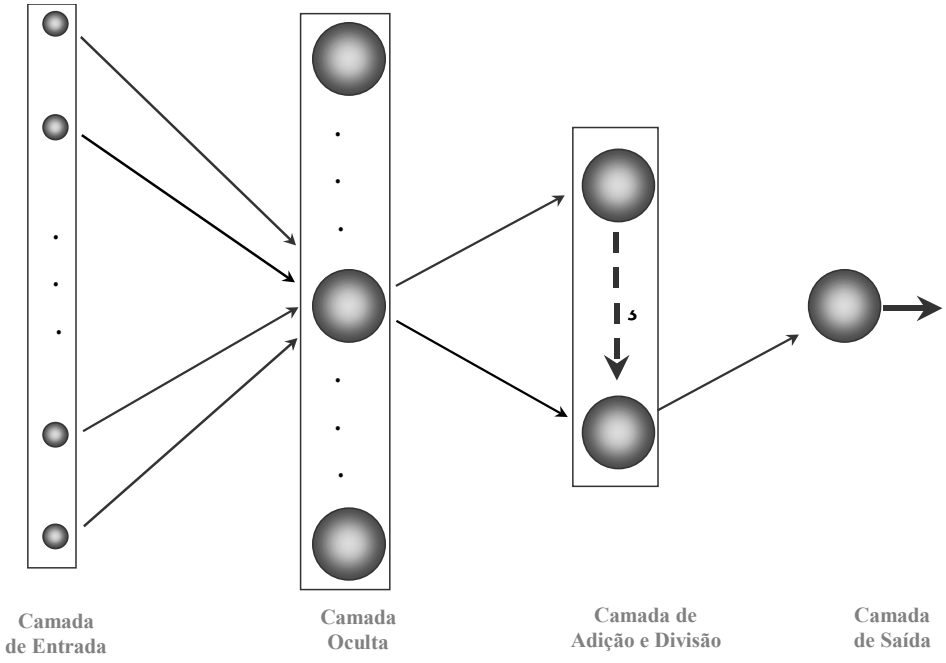


Figura 12 – Rede Feedforward Multicamadas

As principais vantagens das GRNN são:

→ aprendizado rápido, portanto, baixo tempo de treinamento;

→ manuseia tanto dados lineares como não-lineares;

→ converge para uma superfície ótima de regressão quando há muitas amostras;

→ pode ser usada com dados escassos;

→ a ocorrência de *overtraining* é menos provável, pois somente um parâmetro é ajustado;

→ adicionando novos exemplos ao conjunto de treinamento, o modelo não necessita de um novo aprendizado.

Dentre as desvantagens pode-se citar:

→ uso intensivo de memória, pois requer que todos os exemplos de treinamento estejam armazenados para uso futuro, isto é, para fazer predição;

→ utiliza todos os dados de entrada, mesmo os irrelevantes e redundantes;

→ para que o treinamento seja feito em um curto intervalo de tempo, é requerido muitos conjuntos de treinamento.

B) Treinamento

Além da arquitetura de uma Rede Neural, é fundamental entender como é feita a atribuição dos pesos às sinapses. Para tal fim, é usualmente utilizada uma metodologia de treinamento da rede, isto é, os valores das sinapses são modificados aos poucos, visando a minimizar os erros e otimizar a saída da rede. O paradigma utilizado para o treinamento de uma Rede Neural Artificial é o aprendizado do cérebro humano.

Através dos algoritmos de treinamento é que se definem os pesos sinápticos e o processo de aprendizagem. Este processo, independentemente do tipo de algoritmo

utilizado, segue uma sequência de eventos. A princípio, a Rede Neural é estimulada pelo meio; sofre então uma mudança em seus parâmetros como resposta ao estímulo e, finalmente, responde ao meio com uma nova condição obtida através da mudança ocorrida na sua estrutura interna.

A arquitetura da rede define, dentre outros parâmetros, a que tipo de treinamento a rede será submetida, capacitando-a a resolver o problema. Os algoritmos podem ser divididos em três classes: aprendizado supervisionado, aprendizado não-supervisionado e aprendizado por reforço.

O treinamento supervisionado utiliza um agente externo – supervisor – para indicar à rede a resposta desejada para o padrão de entrada. Através do erro, que é a diferença entre os valores esperados e os valores obtidos, o agente externo ajusta os parâmetros da rede. Este ajuste é feito até que o erro seja minimizado, passando a não existir mais ou atingindo um valor considerado satisfatório. A partir deste momento, diz-se que a rede adquiriu conhecimento e apresenta-se treinada. Na figura 13, está representado esquematicamente, em um diagrama de blocos, o treinamento supervisionado.

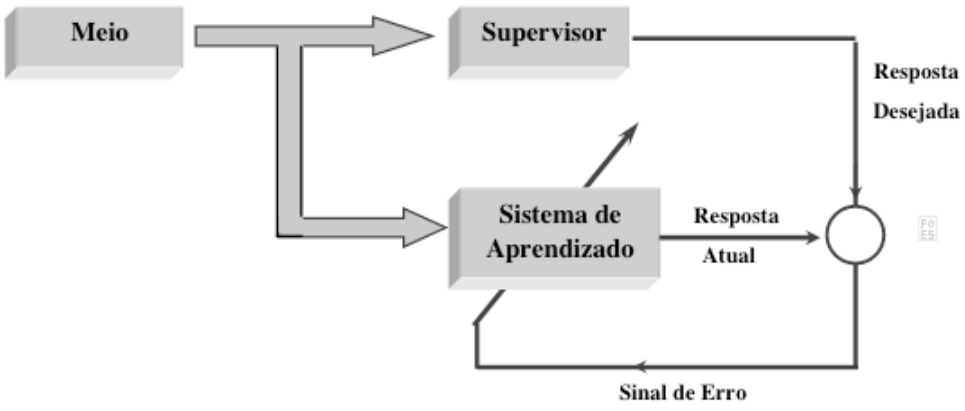


Figura 13 – Diagrama de Blocos do Treinamento Supervisionado

Dentre os algoritmos de treinamento supervisionado, pode-se citar como exemplo o do Erro Médio Quadrático e a generalização do mesmo, o *backpropagation*. Entre os modelos que utilizam este tipo de treinamento, tem-se o dos Perceptrons Multicamadas e as Redes de Base Radial.

O treinamento não-supervisionado não apresenta uma saída-alvo, portanto a própria rede deverá ser capaz de extrair as características relevantes dos impulsos, classificando-os em grupos pré-existentes.

Dado um impulso externo, a rede deverá ser capaz de, extraídas as características relevantes, agrupá-las de acordo com as semelhanças em uma classe já criada. Caso nenhuma classe semelhante seja encontrada, o sistema deverá então criar uma nova classe para o padrão de entrada apresentado.

O ajuste dos pesos é feito de acordo com um conjunto de regras pré-estabelecidas até que se chegue a uma configuração final. O diagrama de blocos do aprendizado

não-supervisionado é apresentado na figura 14.

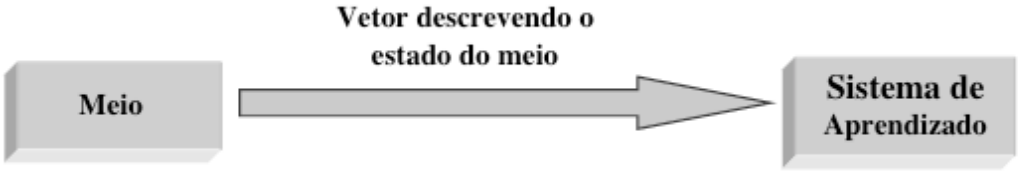


Figura 14 – Diagrama de Blocos do Treinamento Não-Supervisionado.

Pode-se citar como algoritmo de treinamento não-supervisionado o de Aprendizado Competitivo, e como modelo de Rede Neural Artificial que segue esse tipo de treinamento têm-se os Mapas Auto-Organizáveis de *Kohonen* e os Modelos ART de *Grossberg & Carpenter*.

O Treinamento por Reforço pode ser considerado uma variante do aprendizado supervisionado, no qual não se dispõe de respostas corretas, mas pode-se saber se as respostas que a rede produziu são corretas ou não.

Neste algoritmo, um "crítico" irá observar o funcionamento do sistema. Caso as respostas a determinados impulsos sejam satisfatórias, deve-se reforçar as conexões que levam a estas respostas e, caso contrário, estas conexões devem ter um menor peso.

É um método baseado em tentativa e erro, pois os ajustes dos pesos a serem tomados irão depender unicamente das respostas produzidas pelo sistema durante o treinamento. O que o diferencia do treinamento supervisionado é que o supervisor sabe exatamente como ajustar os pesos no caso de erro. Este tipo de treinamento é representado em um diagrama de blocos na figura 15.

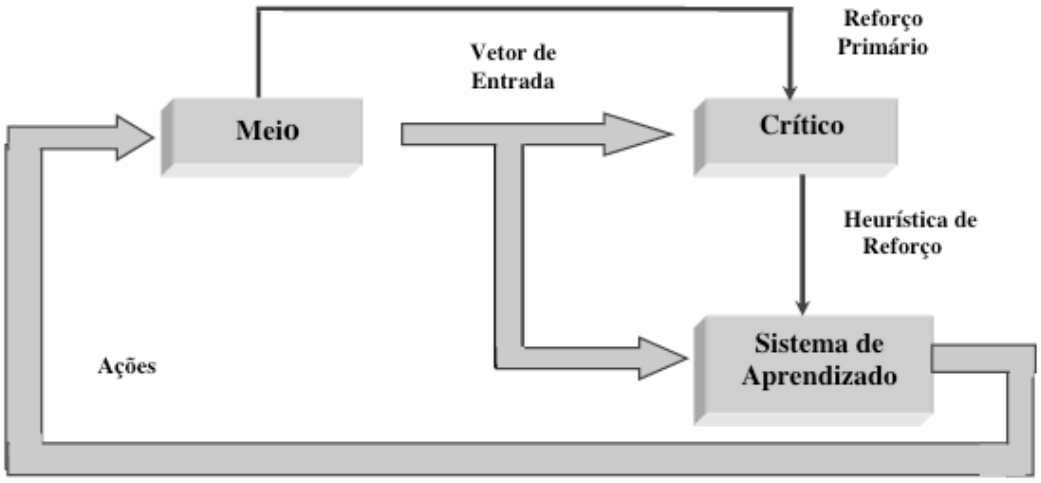


Figura 15 – Diagrama de Blocos do Treinamento por Reforço

C) Aprendizado

Para uma Rede Neural Artificial, o conceito de treinamento diferencia-se do conceito de aprendizado, pois este último está associado a uma tarefa que a rede está executando em função do treinamento, da sua arquitetura e da sua topologia. Já o treinamento é o processo de ensinar a RNA.

É possível dividir o aprendizado em diversos grupos. Entre estes grupos pode-se destacar o Aprendizado Auto Associativo, no qual a rede aprende a associar padrões de mesma natureza, isto é, as entradas e saídas correspondem às mesmas variáveis. Assim, ao receber uma determinada entrada, ela responde invocando a entrada mais parecida que estiver presente no conjunto de treinamento; e o Aprendizado Hetero Associativo, no qual a rede associa padrões de natureza diferentes. As respostas são dadas em função do mapeamento entre as entradas e saídas que a rede aprendeu. Este tipo de aprendizado é ainda classificado em aproximadores (a rede interpola ou extrapola uma saída correspondente à nova entrada) e em classificadores (a rede separa em classes as entradas, em função de semelhanças e diferenças).

D) Backpropagation

Este é um algoritmo de treinamento supervisionado para redes do tipo *feedforward* que tenham neurônios com qualquer função de ativação que seja derivável. Este algoritmo permite modificar aos poucos os valores das sinapses de modo a otimizar a saída da rede.

Partindo do objetivo básico de uma Rede Neural, que é realizar uma função não-linear relacionando P pares entrada-saída ($\underline{x}^k$, $\underline{y}^k$), tal que a equação (IV) seja verdadeira.

$$\underline{y}^k = f(\underline{x}^k) \quad \forall\, k = 1, 2, ..., P \qquad \text{(IV)}$$

Porém, independente da aplicação, na maioria das vezes a Rede Neural realiza uma aproximação, como mostrada na equação (V).

$$\underline{\tilde{y}}^k = f(\underline{\tilde{x}}^k) \cong \underline{y}^k \qquad \text{(V)}$$

Quanto menor o erro decorrente desta aproximação, melhor a rede estará mapeando o problema; portanto, o que a rede deverá fazer é minimizar o erro para todos os pares entrada-saída, através de uma possível Função Custo (F_0) que otimizará a aproximação.

A Função Custo é o valor esperado do erro quadrático ε^k entre a saída ideal $\underline{y}$ e a saída real $\underline{\tilde{y}}$ para todos os pares ($\underline{x}^k$, $\underline{y}^k$).

$$F_0(\underline{w}) = E\left(\varepsilon^k\right) = E\left\{\left\|\underline{y} - \underline{\tilde{y}}\right\|^2\right\}$$

$$F_0(\underline{w}) = \frac{1}{P}\sum_{k=1}^{P}\sum_{l=1}^{M}\left(y_l^k - \tilde{y}_l^k\right)^2 \qquad \text{(VI)}$$

$$F_0(\underline{w}) = \frac{1}{P}\sum_{k=1}^{P}\sum_{l=1}^{M}\left(\varepsilon_l^k\right)^2$$

onde E (1) é o valor esperado de 1 e ε_l^k é o erro linear na saída l para o par ($\underline{x}^k$, $\underline{y}^k$).

$$\varepsilon_l^k = \left(y_l^k - \tilde{y}_l^k\right) \qquad \text{(VII)}$$

Deve-se agora encontrar o valor do vetor de sinapses, $\underline{w}$, que minimize F_0 na equação (VI). Uma forma de se fazer isto é através do Método do Gradiente Descendente, deslocando o vetor de sinapses na mesma direção e sentido contrário ao gradiente F_0. Para tanto, a cada passo do treinamento cada sinapse w_{ij} deve sofrer um acréscimo, como mostrado na equação (VIII), onde $\alpha > 0$ é o passo do treinamento e ε^k é o erro quadrático para um par ($\underline{x}^k$, $\underline{y}^k$).

$$\Delta w_{ij} = -\alpha\frac{\partial F_0(w)}{\partial w_{ij}} = -\alpha\frac{\partial E\left(\varepsilon^k\right)}{\partial w_{ij}} \qquad \text{(VIII)}$$

Seja uma entrada x_k em uma rede, como a representada na figura 16 - na qual a sinapse w_{ij} se apresenta em destaque – que produz sinais internos v_j^k e v_i^k e uma saída $\underline{\tilde{y}}^k$. O caminho g_{li}^k é o ganho para pequenos sinais da saída da sinapse w_{ij} à saída $\tilde{y}_l$.

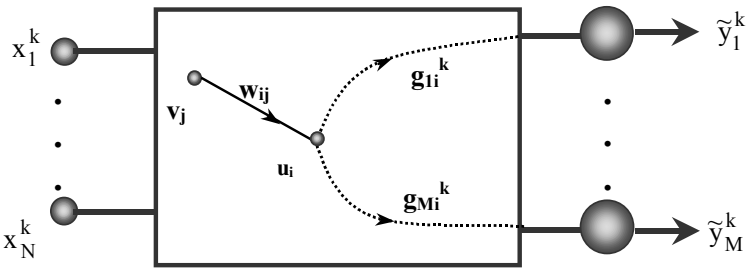


Figura 16 – Sinal Feedforward.

O valor de g_{li}^k pode ser obtido diretamente por inspeção a partir dos ganhos das sinapses e dos neurônios. Como a saída ideal (y_l^k) independe das sinapses (w_{ij}) e $u_i^k = w_{ij}\, v_j^k + c$, onde c é constante com w_{ij}, após alguma manipulação algébrica chega-se à equação (IX)

$$\frac{\partial \varepsilon^k}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_{l=1}^{M} \left(y_l^k - \tilde{y}_l^k \right)^2 = -2\, v_j^k\, \varepsilon_i^k \qquad \text{(IX)}$$

onde

$$\delta_i^k = \sum_{l=1}^{M} g_{li}^k\, \varepsilon_l^k \qquad \text{(X)}$$

δ_i^k é o somatório dos erros da saída (ε_l^k) ponderados pelos ganhos g_{li}^k dos caminhos da extremidade da sinapse w_{ij} até a respectiva saída. Como pode ser visto na figura 17.

Se o erro encontrado for retropropagado (termo que foi derivado do nome do algoritmo *backpropagation*) pela rede, isto é, se o erro for propagado a partir da camada de saída até a camada de entrada, ela tentará estabelecer o quanto cada sinapse contribuiu para o erro, e este será usado para ajustá-las.

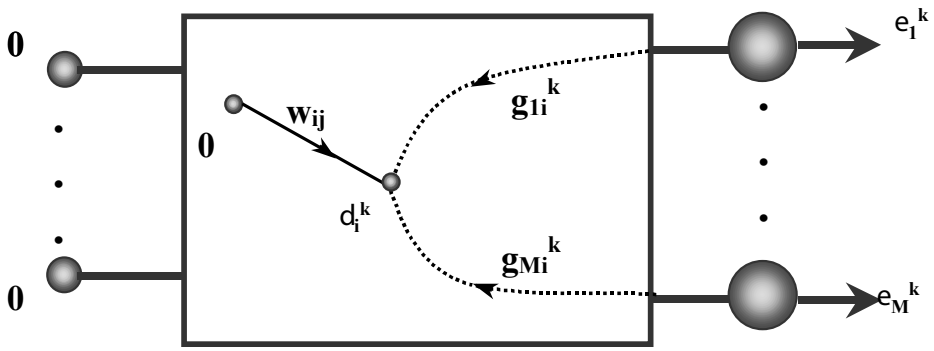


Figura 17 – Error Backpropagation.

E) Correção de Pesos

A correção dos pesos em uma rede *feedforward* é feita através de um ciclo onde os pares entrada-saída do conjunto de treinamento são apresentados no processo de forma a permitir a aprendizagem. Podem ser executados das seguintes maneiras principais:

a. Batelada: Neste processo, todas as P entradas (ou um número delas estatisticamente representativos) são apresentadas à rede; o seu erro médio é calculado e somente após esse cálculo é dado o passo para a correção dos pesos (equação (XI)), o que torna este processo muito lento e, por isso, pouco utilizado.

$$w_{ij}^{novo} = w_{ij}^{anterior} + \Delta w_{ij} \qquad \text{(XI)}$$

b. Regra Delta Modificada: Para acelerar o processo anterior, reduz-se o número de pares entrada-saída usados para tomar uma média. No caso extremo, a média é aproximada pelo valor instantâneo, e a cada par ($\underline{x}^k$, $\underline{y}^k$) a sinapse w_{ij} é atualizada pela equação (XII) aplicada à equação (XI):

$$\Delta w_{ij} = \alpha \; 2 \; v_j^k \; \delta_i^k \qquad \text{(XII)}$$

Isto corresponde a tomar o gradiente do erro quadrático instantâneo, para a entrada k, como uma estimativa do gradiente do erro médio quadrático. Se α for suficientemente pequeno, o valor de w_{ij} permanece “constante” durante vários passos de treinamento. A figura 18 representa esquematicamente como calcular a Regra Delta.

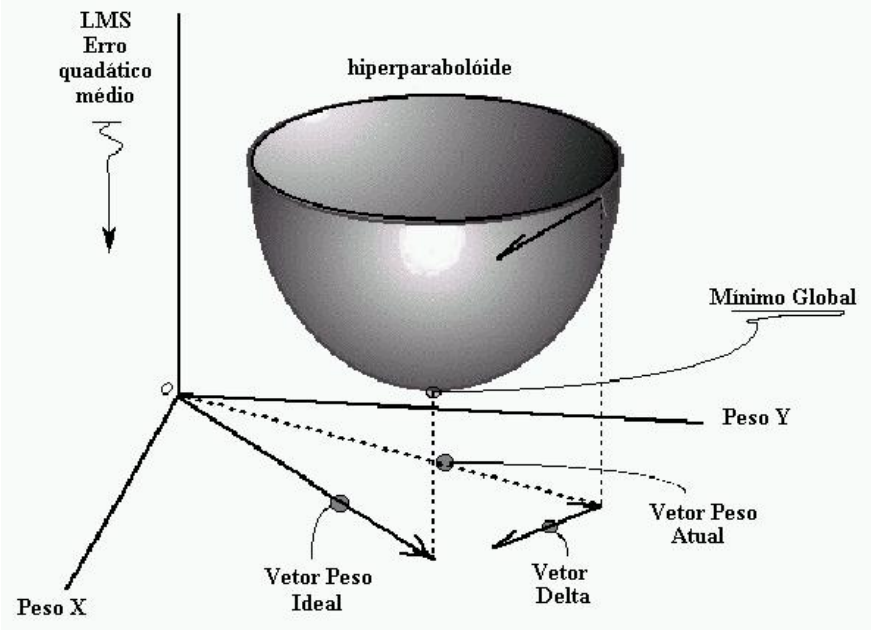


Figura 19 – Regra Delta.

Este processo é equivalente ao de Batelada pois a soma dos diversos Δw_{ij} é equivalente ao obtido pela média dos gradientes do erro quadrático instantâneo. A vantagem deste processo é não necessitar de memória para extrair a média dos dados e o valor de α não precisa ser tão pequeno. A desvantagem é que a convergência pode ser mais oscilatória do que o processo Batelada.

c. Momento: A atualização é feita passo a passo como na Regra Delta, mas o valor de Δw_{ij} é uma ponderação entre o calculado no passo atual e os calculados anteriormente, como no Batelada, como pode ser visto matematicamente na equação (XIII).

$$\Delta w_{ij} = \beta\ \Delta w_{ij}^{anterior} + (1-\beta)\ \Delta w_{ij}^{calculado} \qquad \text{(XIII)}$$

Tipicamente β = 0,9. Este é um processo intermediário entre a Regra Delta Modificada e a Batelada, que tenta reunir a rapidez e simplicidade do primeiro com a maior estabilidade na convergência do segundo.

F) Algoritmo Backpropagation

Dado um conjunto de treinamento com L pares entrada-saída, uma rede de M camadas é treinada pelo seguinte algoritmo:

1. As variáveis de entrada e saída são normalizadas, fazendo-se uma transformação algébrica para que variem no intervalo –1 e 1;
2. Escolhe-se para inicializar um conjunto de parâmetros W^m e b^m, para m = 1, ...M Este valor geralmente varia entre – 0,1 e 0,1;
3. Escolhe-se uma taxa de aprendizado η. Se este valor for grande, a rede

pode divergir e, se for pequeno, pode tornar o processo muito lento. Um valor típico fica entre 0,05 e 0,1. Uma possibilidade é iniciar com um valor maior no começo, para acelerar a convergência, e menor ao final, para garantir um ajuste fino;

4. O número de camadas é inicialmente definido como um. Se o problema não chegar a uma solução, opta-se por duas camadas;
5. Para a camada de saída, considera-se uma regra: se a saída for contínua, usa-se a função linear, e, se a saída for lógica, a função tangente hiperbólica;
6. Estabelece-se algum critério de parada, seja este um número máximo de iterações K_{max} ou que o valor esperado atinja um limite máximo pré-estabelecido $E(w) < \lambda$;
7. Calculam-se os erros de saída, da camada de saída e das camadas subsequentes;
8. Atualizam-se as sinapses mediante o ganho obtido;
9. Recalcula-se o vetor sinapse e retorna-se ao passo 7 até satisfazer o critério de parada.

G) Problemas de Treinamento

A. *Mínimos Locais*

O treinamento é um processo de otimização por gradiente em uma superfície, como a da figura 19, e apresenta problemas inerentes a este processo. Um dos problemas é que, como a função custo não é quadrática, podem ocorrer mínimos locais.

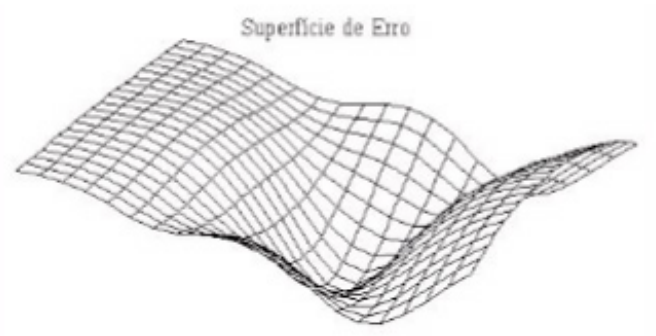


Figura 19 – Exemplo de Superfície de Erro.

Nesta superfície existem “vales” com menor profundidade que outros; estes são os mínimos locais, e não representam a melhor solução (Figura 20), porém o algoritmo *backpropagation* pode permitir a entrada em um mínimo local, não permitindo que se encontre a melhor solução do problema mapeado pela rede. Para resolver este problema, pode-se usar um tipo de treinamento chamado *simulated annealing*, que é muito mais complexo e lento que o método convencional do gradiente descendente.

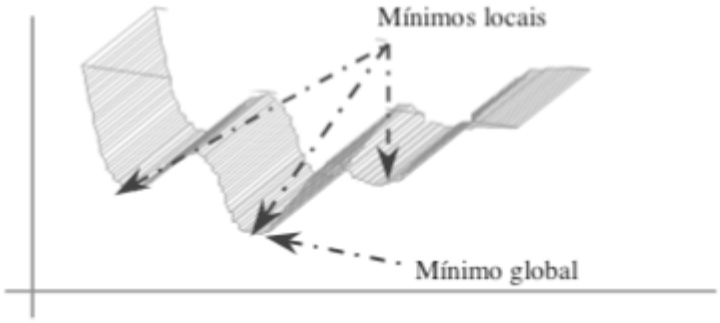


Figura 20 - Mínimos Locais de uma Função.

B. Paralisia

A forma da função $F_0(\underline{w})$ depende da função a minimizar. As superfícies de erros sigmoidais podem apresentar regiões com pequenos declives, isto é, um gradiente de F_0 muito pequeno. Na figura 21, se w for deslocado para uma região de gradiente pequeno, o treinamento fica praticamente paralisado, pois como Δw_{ij} é proporcional ao gradiente ficará muito pequeno. Soluções adotadas para sinalizar ou evitar a paralisia são heurísticas de verificação dos valores de y_i ou u_i muito elevados ($u_i > 5$) ou então um limite máximo para os valores de w_{ij}.

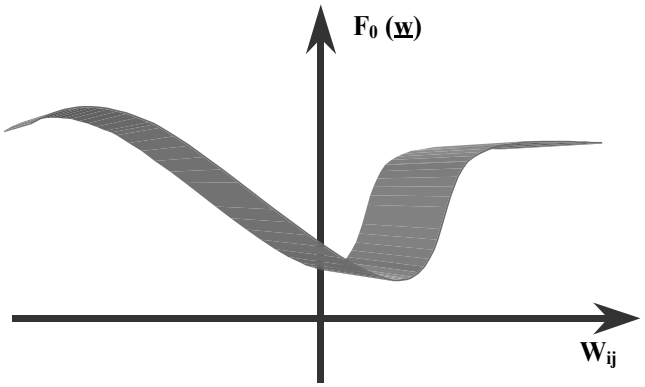


Figura 21 – Função Erro.

C. *Passo de Treinamento*

É dado pelo "tamanho do passo" (α) que o algoritmo utiliza para caminhar pela superfície. Valores muito pequenos de α tornam o treinamento longo, e valores grandes podem provocar divergência do mesmo. Para a maioria dos casos, se $\underline{x}^k$ tem suas componentes normalizadas ($|x_i^k| \leq 1, \forall\, i, k$) e o maior número de sinapses que alimenta um neurônio é N_{max} o, treinamento converge se a equação (XIV) for verdadeira.

$$\alpha \leq \frac{1}{N_{max}^2} \qquad \text{(XIV)}$$

Isto é considerado um limite muito conservador; tipicamente α é escolhido no entorno de 0,1.

Outro problema é que, se o erro mínimo no fim do treinamento ainda for elevado os acréscimos às sinapses serão grandes e ele oscilará em torno de um valor ótimo. Isto pode ser contornado iniciando o treinamento com um valor de α maior e diminuí-lo ao progredir o treinamento.

D. *Pesos Iniciais*

Os valores iniciais das sinapses devem ser pequenos, para evitar a paralisia já nos primeiros passos de treinamento, devido a valores elevados de u_i. Portanto, se um neurônio i é alimentado por N sinapses, o valor inicial de u_i pode ser limitado a $|u_i| \geq 1$ se cada sinapse do neurônio i for limitada a $|w_{ij}| \geq 1/N$.

E. Testes realizados pela rede

Uma Rede Neural Artificial realiza tanto o treinamento como o teste com os pares entrada-saída ($\underline{x}^k$, $\underline{y}^k$) disponíveis. Estes pares são divididos em dois conjuntos: o conjunto de treinamento e o conjunto de teste. A rede é treinada com o conjunto de pares de treinamento, e a performance obtida é medida com o conjunto de pares de teste.

Isto é necessário porque um treinamento prolongado demais (*overtraining*) leva a rede a uma super-especialização – sobretudo se houver poucos pares entrada-saída disponíveis para o treinamento – que piora a performance quando os dados de teste são apresentados.

A figura 22 apresenta a evolução típica do erro medido no conjunto de treinamento (linha tracejada) e no conjunto de teste (linha cheia) em função do número de passos de treinamento. É claro que o treinamento deve ser parado em $n = n_c$.

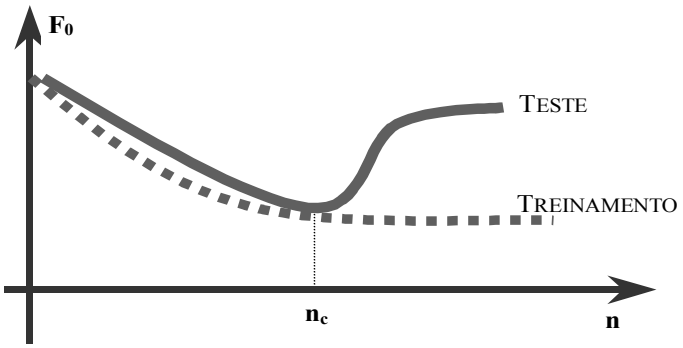


Figura 22 – Erro no Treinamento.

H) Treinamento das Redes GRNN

A topologia de uma GRNN consiste de 3 camadas: uma escondida, uma que realiza adição e divisão, e a de saída. O treinamento é rápido e feito em um só passo,

pois os vetores de entrada são simplesmente copiados para a camada escondida, logo, cada neurônio é representante de um vetor do conjunto de treinamento apresentado à rede. Quando um conjunto de valores desconhecidos é apresentado, a distância entre estes e os do conjunto de treinamento é calculada, computada e passada para a camada seguinte.

Na camada de adição e divisão existem 2 nós, chamados, por exemplo, de A e B. O nó A computa a soma dos pesos referentes a cada neurônio representante dos vetores conhecidos, e o nó B, simplesmente, computa a soma das distâncias. Os pesos destas médias calculadas decaem exponencialmente com a distância entre os pontos. O nó da camada de saída somente apresenta o valor de B dividido por A, fornecendo então a predição.

Uma GRNN pode ser melhor entendida como uma interpolação da função que deve ser ajustada. Esta é chamada de fator de *smoothing* e é o único parâmetro que sofre modificações, permitindo que a GRNN interpole entre os padrões desconhecidos apresentados e os do conjunto de treinamento. A otimização deste fator é considerada crítico para o desempenho da GRNN e é usualmente encontrado por ajustes interativos de validação.

I) Determinação Semi-Empírica de Arquiteturas

Os sistemas baseados em Redes Neurais Artificiais dependem fortemente da topologia das redes (tamanho, estrutura, conexões), assim como de seus parâmetros (taxa de aprendizado, momento). Como resultado, a determinação da arquitetura da rede afeta muito o seu desempenho.

Se a rede for subdimensionada, ela apresentará pouca capacidade de generalizar ou mesmo nem aprenderá, isto é, o erro mínimo na saída permanecerá muito grande; se for superdimensionada, aumentará desnecessariamente o tempo de treinamento ou necessitará de máquinas com maior capacidade de processamento.

Portanto, é grande a dificuldade de projetar redes neurais eficientes, mas existem algumas técnicas que envolvem conhecimentos heurísticos que podem ajudar.

A princípio, sabe-se somente que a última camada deve ter M = dimensão ($y_{saída}$) neurônios, isto é, o número de neurônios da camada de saída é o mesmo número de variáveis que serão preditas. Isto não é suficiente, pois só uma camada não mapeia problema.

O dimensionamento da rede depende do tipo de função a aproximar, e a forma mais usada para realizá-lo ainda é tentativa-e-erro. Inicia-se com 1 camada, se o erro obtido for muito grande, passa-se para uma rede de 2 camadas.

O número de neurônios na camada oculta deve estar, à princípio, entre o número de vetores de entrada e de saída. Se a rede aprender, pode-se tentar reduzir o número de neurônios na camada intermediária, caso contrário, aumentá-los. Se com duas camadas o aprendizado ainda não foi bem-sucedido, passa-se para três camadas e

assim sucessivamente. Existem, entretanto, alguns algoritmos para dimensionamento.

Se a rede apresenta a capacidade de realizar a função requerida ela pode ser otimizada por *Algoritmos Destrutivos*. Este diminui o número ou o módulo das sinapses. Para reduzir um neurônio, basta eliminar todas as suas sinapses.

Outra maneira é utilizar *Algoritmos Construtivos*, partindo de uma rede mínima e ir aumentando-a até que realize a função desejada. Este tipo algoritmo é mais empregado em classificadores.

Para iniciar o treinamento, sugere-se que os parâmetros sejam: pesos devem estar entre ± 0,3; taxa de aprendizado no entorno de 0,1; e momento em 0,9.

MAPAS AUTO ORGANIZÁVEIS: SOM

O Self-Organizing Map (SOM), ou Mapas Auto Organizáveis, foram desenvolvidos por Teuvo Kohonen em 1982. São redes com aprendizado não-supervisionado e baseada em Aprendizagem Competitiva.

O desenvolvimento dos SOM´s como modelo neural foi motivado pela característica da organização do cérebro humano em muitas regiões, de tal forma que entradas sensoriais distintas são representadas por mapas computacionais topologicamente ordenados. Por exemplo, entradas sensoriais tácteis, visuais e acústicas são mapeadas em diferentes áreas do córtex cerebral, de uma forma topologicamente ordenada.

Baseado nesta neurofisiologia, as Redes SOM ou Redes de Kohonem, se utilizam do seguinte paradigma: quando um neurônio é excitado, ao seu redor, uma área entre 50 e 100 µm, também sofre excitação e, a partir desta distância sofre inibição, de forma a impedir a propagação do sinal a áreas não relacionadas, conforme figura 23.

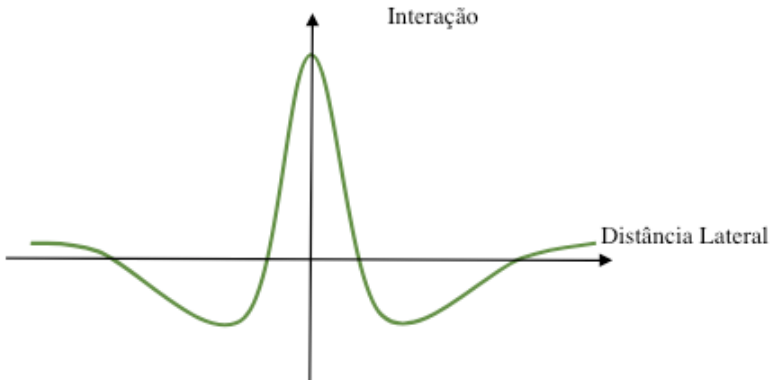


Figura 23 – Interação lateral entre os neurônios.

Na RNA este feedback lateral é modelado através de uma função conhecida como chapéu mexicano. Sabendo-se que cada neurônio influência o estado de ativação de seus vizinhos de maneiras diferentes:

✓ Excitatória: se os vizinhos estão próximos a ele;

✓ Inibitória: se os vizinhos estão fora do raio de vizinhança;

✓ Levemente excitatória: quando os vizinhos estão dentro de uma área menos de vizinhança.

A Figura 24 apresenta dois exemplos de funções Chapéu Mexicano:

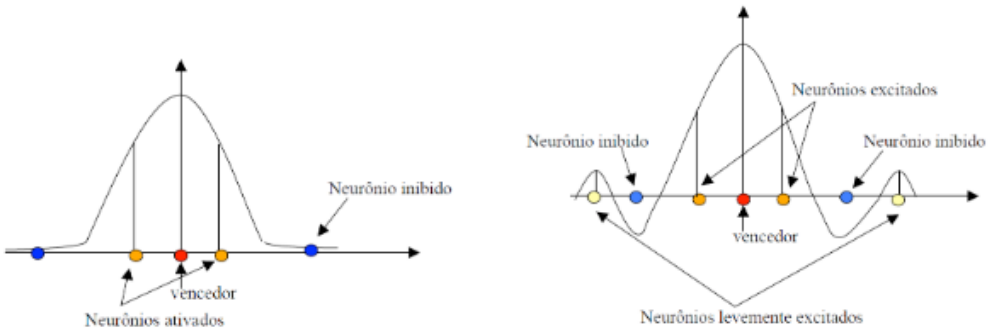


Figura 24 – Exemplos de Funções Chapéu Mexicano.

J. Aprendizagem Competitiva

As redes SOM são baseadas no aprendizado competitivo. Os neurônios de saída da RNA competem entre si para serem ativados, onde apenas um neurônio de saída (ou um neurônio por grupo) está “ligado” a qualquer instante. Um neurônio de saída que vence tal competição é chamado neurônio vencedor (*winner-takes-all neuron*).

Para induzir tal tipo de competição entre os neurônios de saída usa-se conexões inibitórias laterais entre eles, através de caminhos de realimentação negativa.

Os neurônios em uma rede SOM são colocados nos nós de uma treliça (*lattice*) de uma ou duas dimensões, normalmente.

Os neurônios se tornam seletivamente ligados aos padrões ou classes de entrada, que são considerados estímulos, ao longo de um processo competitivo de aprendizado. A localização destes neurônios denominados vencedores se torna ordenada entre si de tal forma que um sistema de coordenadas significativo é criado na treliça, para diferentes características de entrada.

Um SOM é, portanto, caracterizado pela formação de um mapa topográfico dos padrões de entrada, no qual as localizações espaciais (ou coordenadas) dos neurônios na treliça são indicativas de características estatísticas intrínsecas contidas nos padrões de entrada.

O mapa computacional é como uma matriz de neurônios que operam nos sinais que transportam informação sensorial, em paralelo. Desta forma, os neurônios transformam sinais de entrada em uma distribuição de probabilidade codificada na região que representa os valores computados de parâmetros por regiões de máxima atividade relativa dentro do mapa. A informação assim derivada é tal que pode ser acessada prontamente utilizando esquemas relativamente simples.

O principal objetivo das redes SOM é transformar um sinal padrão de entrada de dimensão arbitrária em um mapa discreto de uma ou duas dimensões, dinâmica e adaptativamente, em uma forma topologicamente ordenada.

Cada padrão de entrada apresentado à rede consiste de uma região localizada de atividade. A localização e natureza de tal região usualmente varia de uma realização de padrão de entrada, para outra. Todos os neurônios na rede devem, portanto, ser expostos a um número suficiente de diferentes realizações dos padrões de entrada,

para garantir que o processo de auto-organização ocorra de forma apropriada.

A formação dos mapas auto organizáveis está representada na Figura 25.

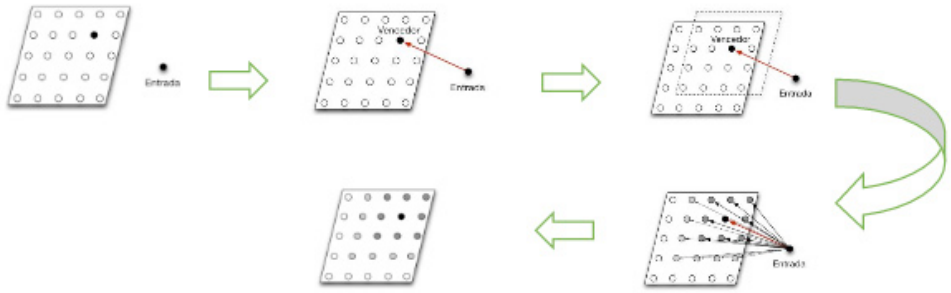


Figura 25 – Mapas Auto Organizáveis.

O algoritmo responsável pela formação do SOM inicia os pesos sinápticos aleatoriamente atribuindo pequenos valores, desta forma, nenhuma ordem prévia é imposta ao mapa de características. Desde que a rede tenha sido adequadamente inicializada, há três processos essenciais envolvidos na formação do SOM: Competição, Cooperação e Adaptação Sináptica.

Na Competição, para cada padrão de entrada, os neurônios da rede computam os seus respectivos valores de uma função discriminante. Esta função provê as bases para a competição entre os neurônios. O particular neurônio com o maior valor de função discriminante é declarado vencedor da competição.

Na Cooperação, o neurônio vencedor determina a localização espacial de uma vizinhança topológica de neurônios excitados, provendo, desta forma, as bases para a cooperação entre tais neurônios vizinhos.

No último mecanismo, a Adaptação sináptica permite aos neurônios excitados aumentar seus valores individuais da função discriminante em relação ao padrão de entrada, através de ajustes adequados aplicados a seus pesos sinápticos. Os ajustes feitos são tais que a resposta do neurônio vencedor à subsequente aplicação de um padrão similar de entrada é realçada.

É uma forma de aprendizado que divide o conjunto de padrões de entrada em grupos inerentes aos dados. Em sua forma mais simples é “vencedor leva tudo”. Os neurônios de saída da RNA competem entre si para se tornar ativos e apenas um neurônio de saída está ativo em um determinado instante.

Há três elementos básicos: (i) neurônios com mesma estrutura, diferente pelos pesos, de forma que tenham respostas diferentes a uma entrada; (ii) um limite imposto sobre a força de cada neurônio; (iii) mecanismo de competição entre neurônios, de forma que um neurônio é vencedor em um dado instante.

Em cada momento o neurônio vencedor aprende a se especializar em agrupamentos de padrões similares e torna-se detector de características para

classes diferentes de padrões de entrada. O número de unidades de entrada define a dimensionalidade dos dados.

K. Mapa Topológico

No caso bidimensional, dois tipos de grade são possíveis: a hexagonal, onde cada neurônio possui 6 vizinhos diretos, ou a retangular, onde cada neurônio possui 4 vizinhos diretos, conforme Figura 26.

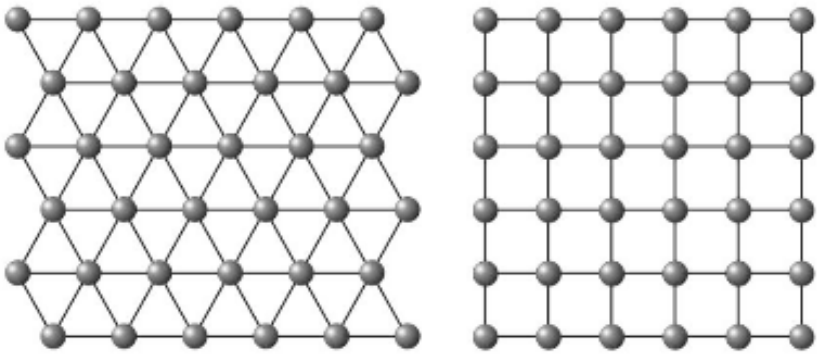


Figura 26 – Mapas topológicos: hexagonal e retangular.

L) SOM – Algoritmo Básico

1. Inicialização

Inicialização aleatória: mais simples e mais utilizada

a. nenhum estado de organização é considerado;

b. durante as primeiras centenas de passos, os vetores peso passam por processo de auto-organização.

Inicialização linear: tenta pré organizar o espaço

c. arranjo de neurônios definido num subespaço linear que corresponda aos *k* auto-vetores da matriz de auto-correlação de *X* que possuem os maiores auto-valores;

d. similar à análise dos componentes principais (PCA).

2. Competição: para cada padrão de entrada, calcula-se a resposta dos neurônios de saída (grade). O neurônio com a maior resposta é o vencedor da competição.

- O critério do melhor casamento (*best matching*) é baseado na maximização do produto interno como na aprendizagem competitiva;

- Matematicamente equivalente a minimizar a distância euclidiana entre **w** e **x.**

3. Cooperação: o neurônio vencedor define uma vizinhança topológica (em função da grade) de neurônios excitados.

- Compreende a definição de uma função de vizinhança *hvj* centrada no neurônio vencedor;

- Define uma região de neurônios cooperativos, que terão seus pesos ajustados juntamente com o neurônio vencedor;

- Há diversas formas de implementar a função de vizinhança, sendo que a mais simples é definir um conjunto *Nv(t)* de níveis de vizinhança ao redor do neurônio vencedor.

4. Adaptação Sináptica: aprendizado em relação ao padrão de entrada. Os pesos do neurônio vencedor, e de sua vizinhança, ficam mais próximos do padrão de entrada.

Durante o processo de aprendizagem os neurônios organizam-se e tornam-se ordenados entre si, especializando-se em detectar determinadas características dos padrões de entrada.

O SOM é, portanto, caracterizado pela formação de um mapa topológico dos padrões de entrada e a localização espacial dos neurônios na grade após o aprendizado podem ser considerados indicativos das características estatísticas contidas nos padrões de entrada.

A tabela 1 apresenta resumidamente exemplos de possíveis áreas de aplicação.

Aplicação	Direta, Multicamada (retropropagada	Hopfield	Boltzmann	SOM (Kohonen)
Classificação	*	*	*	*
Processamento de Imagem	*			
Tomada de decisão	*		*	*
Otimização		*	*	*

Tabela 1: "Áreas de aplicação" por tipos de redes.

A tabela 2 mostra resumidamente exemplos de possíveis áreas de aplicação conforme a estrutura das Redes Neurais Artificiais.

<table>
<tr><th>Estrutura</th><th>Uma Camada
Conexões laterais</th><th>Mapa de vetor topológico</th><th>Duas camadas
Diretas e Reversas</th><th>Múltiplas camadas,
Diretas</th></tr>
<tr><td>Tipo de rede</td><td>Hopfield</td><td>SOM
(Kohonen)</td><td>ART</td><td>MLP's
Rede de Boltzmann</td></tr>
<tr><td rowspan="4">Área de aplicação</td><td rowspan="2">Auto associação</td><td>Auto-associação</td><td rowspan="2">Hetero-associação</td><td>Otimização</td></tr>
<tr><td>Reconhecimento de padrões</td><td>Reconhecimento de padrões</td></tr>
<tr><td rowspan="2">Otimização</td><td>Otimização</td><td rowspan="2">Reconhecimento de padrões</td><td>Compressão de dados</td></tr>
<tr><td>Compressão de dados</td><td>Filtros</td></tr>
</table>

Tabela 2: Áreas de aplicação" de alguns tipos de RNA's, pela sua estrutura.

A figura 27 apresenta um mapa mental com um resumo geral das RNAs.

Figura 27 – Mapa Mental de Redes Neurais Artificiais.

A figura 28 apresenta um mapa mental com o treinamento de uma RNA.

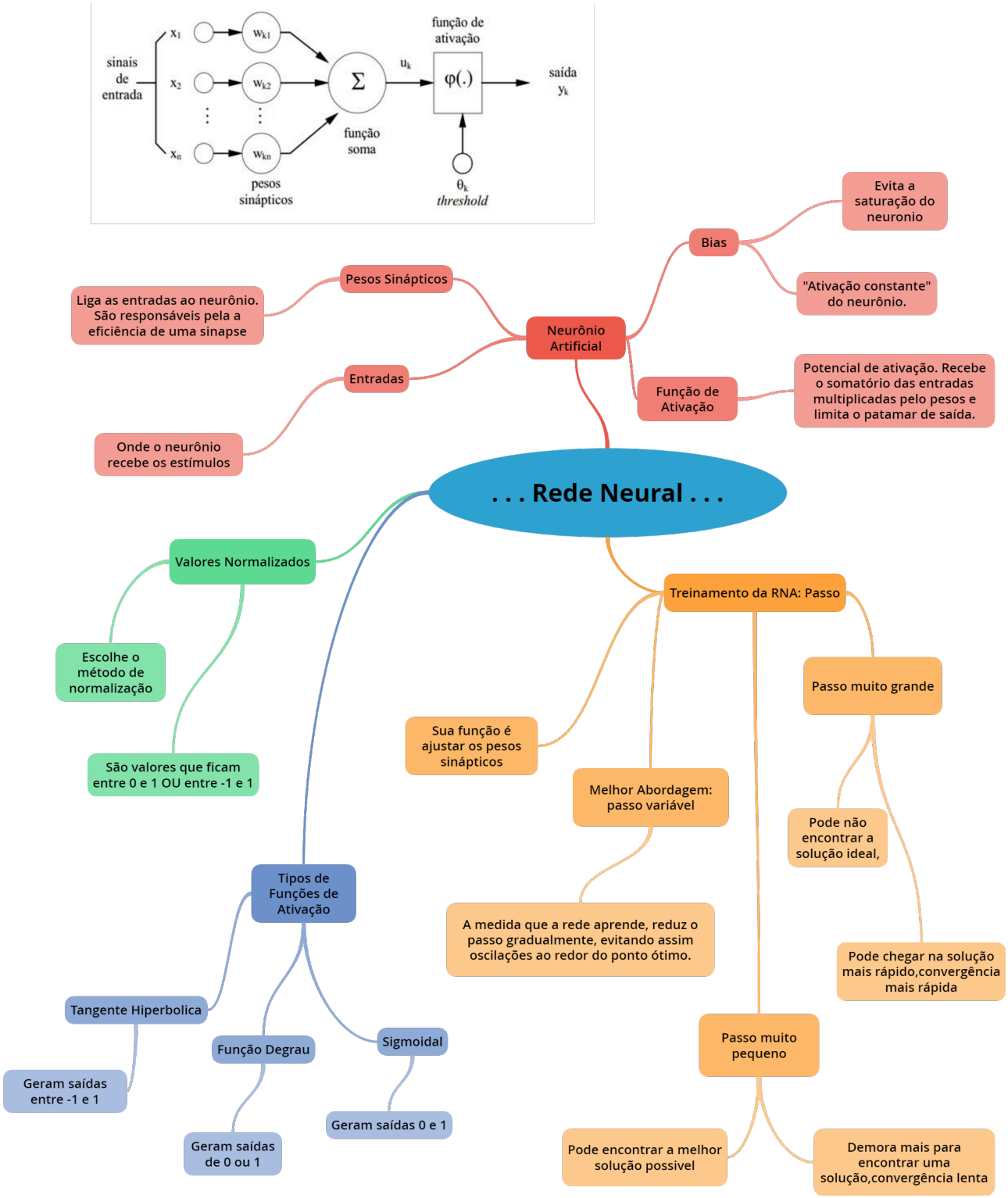


Figura 28 – Mapa Mental do treinamento de uma RNA.

APLICAÇÕES PRÁTICAS E EXERCÍCIOS

M) O Modelo de um Neurônio Artificial

O modelo mais simples de Redes Neurais Artificiais está representado esquematicamente na figura 29.

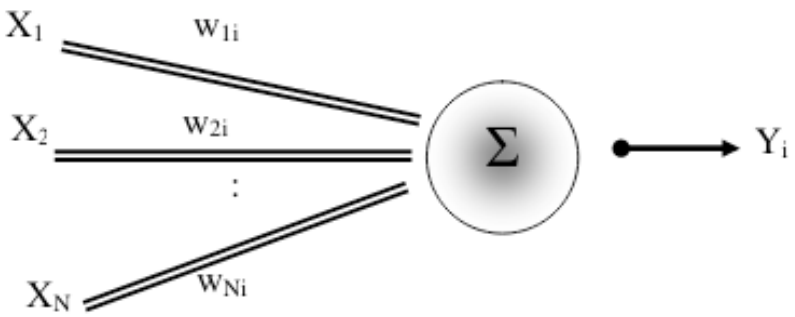


Figura 29 – Modelo inicial do Neurônio Artificial.

O neurônio artificial é formado pelos seguintes componentes:

- Conjunto de entradas X_1, X_2,..., X_N.
- Pesos associados a cada entrada W_1, W_2,..., W_N. Chamamos cada peso de sinapse.
- Um somador Σ para ponderar as respectivas entradas com os pesos associados.
- Uma saída produzida Y_i.

Faz-se necessário modificar a figura 29, adicionando uma função de ativação para restringir a amplitude da saída. Acrescenta-se ainda um parâmetro chamado “bias” que tem o efeito de aumentar ou diminuir a entrada liquida da função de ativação, dependendo se ele é positivo ou negativo. O esquema modificado está representado na figura 30.

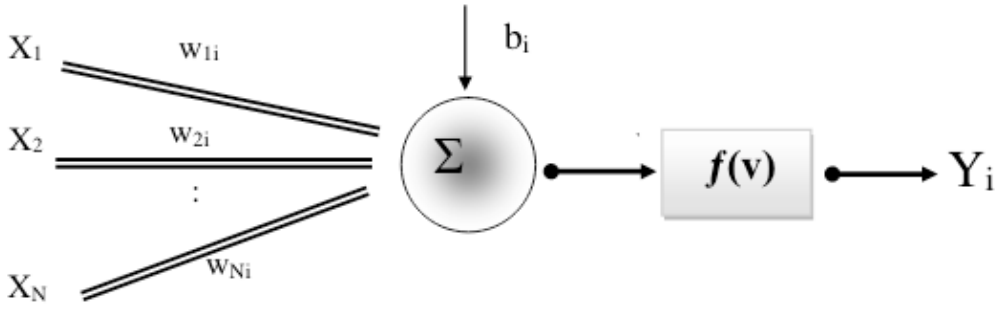


Figura 30 – Neurônio com função de ativação (bias).

Para definir um neurônio usar-se-á um conjunto de equações. Inicialmente a equação XV e XVI:

$$v_i = \sum_{i=0}^{N} w_i \cdot x_i \quad \text{(XV)} \qquad Y_i = f(v_i) \quad \text{(XVI)}$$

O "bias" será considerado como um peso de uma entrada $X_{0,}$ com um valor fixo de +1. Sendo assim, pode-se modificar a equação XV do neurônio, conforme XVII.

$$v_i = \sum_{i=1}^{N} w_i \cdot x_i + b_i \quad \text{(XVII)}$$

Conforme proposto, o neurônio terá seu modelo levemente alterado e passará a ser representado como na figura 31.

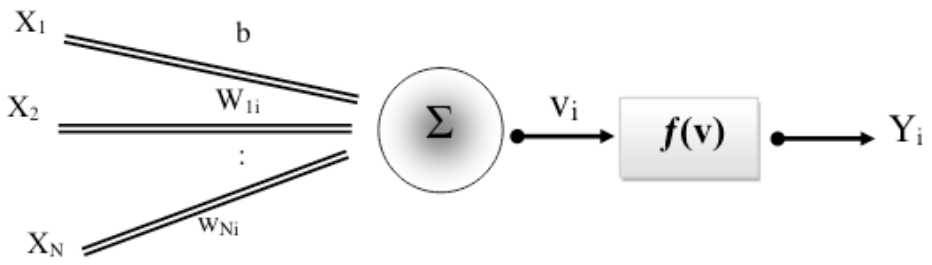


Figura 31 – Neurônio com função de ativação bias, de forma não explícita.

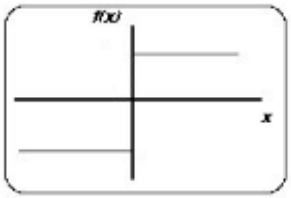
N) Tipos de função de ativação

Neste ponto da modelagem neural, necessita-se uma função de ativação que

limite o valor de saída do neurônio, ou seja, esta função não pode tender ao infinito quando os valores de v_i crescem. Algumas possíveis funções de ativação de um neurônio estão listadas a seguir.

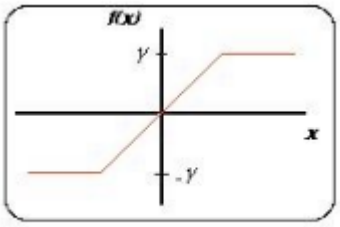
- Função de Limiar

$$f(v) = \begin{cases} 1 \text{ se } v >= 0 \\ -1 \text{ se } v < 0 \end{cases}$$



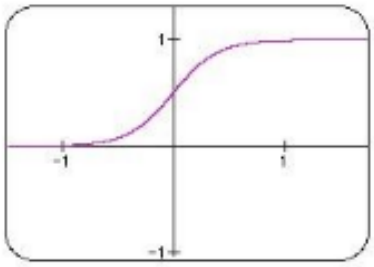
- Função Rampa

$$f(v) = \begin{cases} 1 \text{ se } v >= ½ \\ v \text{ se } -½ < v < ½ \\ -1 \text{ se } v <= -½ \end{cases}$$



- Função sigmóide

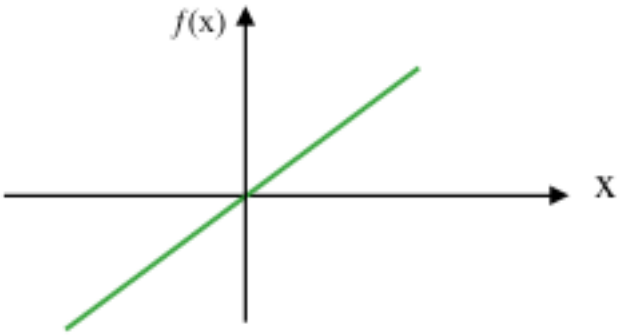
$$f(v) = \tanh(v)$$



Vale ressaltar que existem duas condições necessárias para que uma função matemática possam ser usadas como função de ativação em uma rede neural artificial: a função deve ser contínua; e os limites da função quando x tende a infinito devem ser diferentes de infinitos.

Exemplo 1: Considere a seguinte função matemática f(x) = 2x. Posso usá-la como função de ativação?

Resolução:



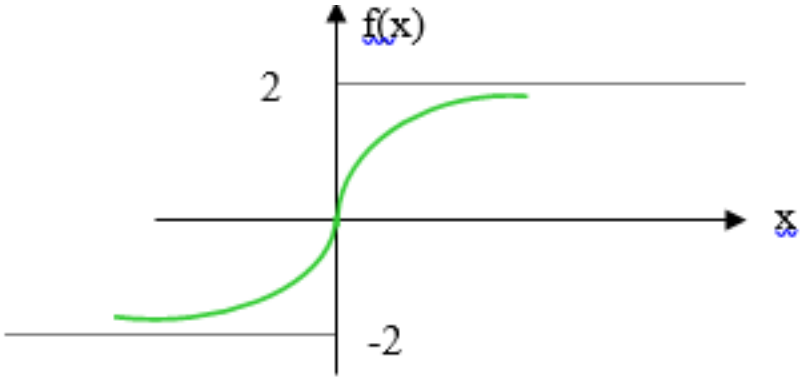
A função f(x) = 2x satisfaz a primeira condição, pois é continua, porém não satisfaz a segunda condição:

$$\lim_{n\to+\infty} 2x = +\infty \qquad e \qquad \lim_{n\to-\infty} 2x = -\infty$$

Sendo assim, **não** é possível utilizá-la como função de ativação.

Exemplo 2: Considere a seguinte função matemática representada pelo gráfico. Esta função pode ser usada como função de ativação ?

Resolução:



1) A função é contínua.

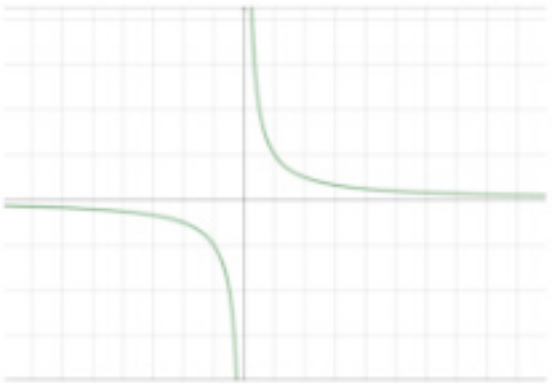
$$\lim_{n\to+\infty} f(x) = 2 \qquad e \qquad \lim_{n\to-\infty} f(x) = -2$$

2) Sim, a função pode ser usada como função de ativação de uma rede neural.

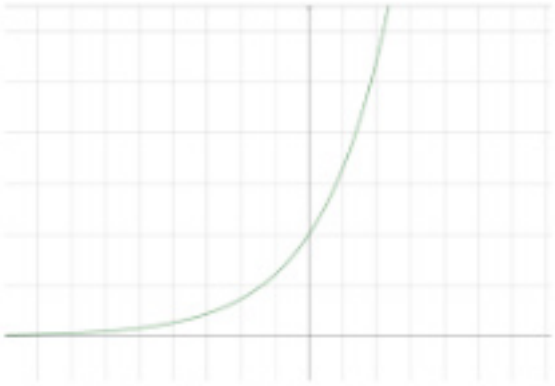
Exercícios Sugeridos

Dadas as funções matemáticas, verifique se pode ser usada como função de ativação e justifique.

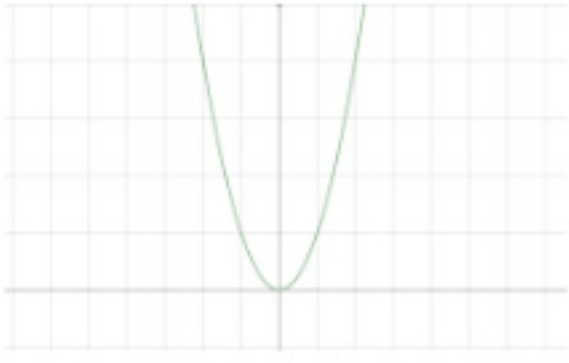
a) $f(x) = 1/x$



b) $f(x) = e^x$



c) $f(x) = x^2$



O) Regra de Hebb

Os resultados obtidos por Hebb motivaram os primeiros métodos de aprendizagem e a ideia básica consiste no fato de que, se um neurônio biológico recebe uma entrada de outro neurônio e ambos se encontram altamente ativados, então, a ligação entre eles é reforçada. Aplicando-se esta observação ao esquema de RNA pode-se dizer que se uma unidade *i* recebe uma entrada de outra unidade *j* altamente ativada, então, a importância desta conexão deve ser aumentada, isto é, o valor do peso entre as

unidades deve ser acrescido. A maneira mais simples de representar matematicamente esta relação é representada pela equação XVIII.

$$W_{ij}(novo) = W_{ij}(antigo) + Y_i Y_j \qquad (XVIII)$$

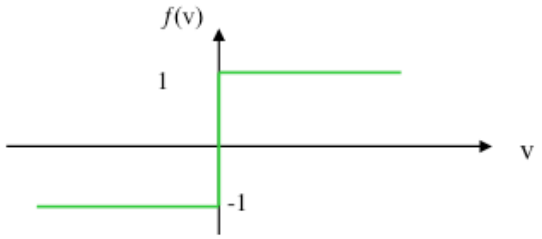
Algoritmo de Hebb

Considere a aplicação do algoritmo de aprendizado de Hebb para treinar uma rede neural que simule o comportamento da função lógica *and* ou (e).

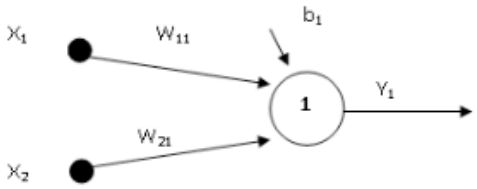
1. As entradas serão apresentadas, um par entrada-saída por vez;
2. Aplica-se a fórmula para cálculo da saída da rede (fórmula XV), obtém-se o valor de saída Y_{obtido};
3. Passar pela função de ativação;
4. Se o valor de Y obtido for igual ao valor de Y desejado (valor real), apresenta-se um novo par entrada-saída, caso contrário, atualiza a sinapse utilizando a fórmula $W_{ji}(novo) = W_{ji}(antigo) + X_j Y_i$;
5. Retorna ao passo, até que não haja mais divergência entre o valor de Y obtido pela rede e o valor de Y real.

Considere este exemplo que utilizará a seguinte função de ativação para treinar uma rede do tipo *and*.

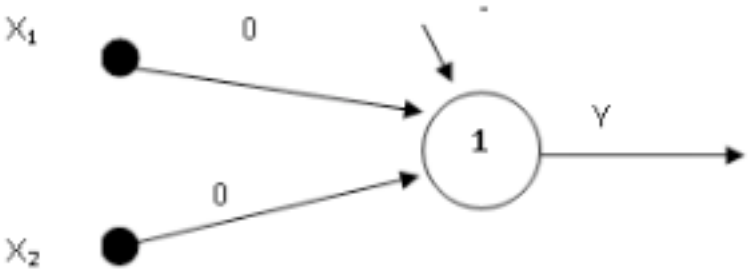
$$f(v) = \begin{cases} 1 \text{ se } v > 0 \\ -1 \text{ se } v <= 0 \end{cases}$$



Entrada 1	Entrada 2	Saída
1	1	1
1	0	-1
0	1	-1
0	0	-1



Passo 1: Inicializar os pesos das sinapses com zero



Para cada par entrada-saída, ajusta-se os pesos das sinapses.

- Primeiro par entrada-saída.

X1	X2	Y
1	1	1

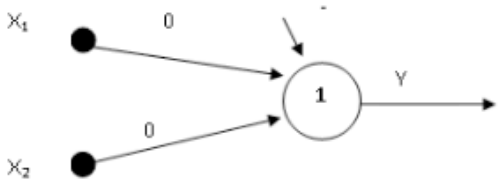
$$v = \sum_{i=0}^{n} x_i$$

$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$

$v_1 = 1*0 + 1*0 + 1*0 = 0$

Pssa pela função de ativação para verificar o valor de saída de Y, ou seja, Y_{obtido}.

$Y_1 = -1 = Y_{real}$



- Segundo par entrada-saída.

X1	X2	Y
1	0	-1

$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$

$v_1 = 1*1 + 0*1 + 1*1 = 2$

$Y_1 = 1$, que é diferente do $Y_{real} = -1$, portanto, ajustar o peso.

$W_{11}(novo) = W_{11}(antigo) + X_1 Y_1$

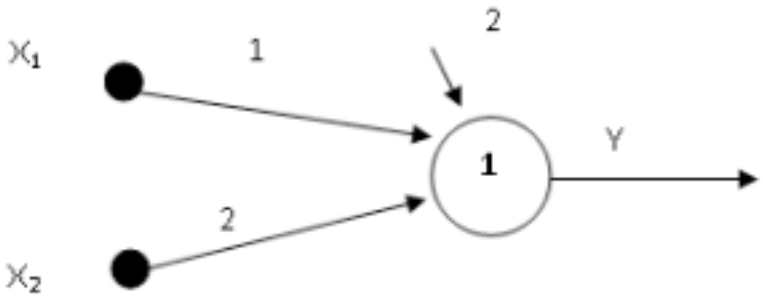
$W_1(novo) = 1 + 1*1 = 2$

$W_{21}(novo) = W_{21}(antigo) + X_2 Y_1$

$W_2(novo) = 1 + (0)*(1) = 1$

$b_1(novo) = b_1\ (antigo) + 1*Y_1$

$b_1(novo) = 1 + 1*(1) = 2$



- Terceiro par entrada-saída.

X1	X2	Y
0	1	-1

$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$

$v_1 = 0*1 + 2*1 + 1*2 = 4$

$Y_1 = 1$, como Y_{real} é diferente, ajustar!

$W_{11}(novo) = W_{11}(antigo) + X_1 Y_1$

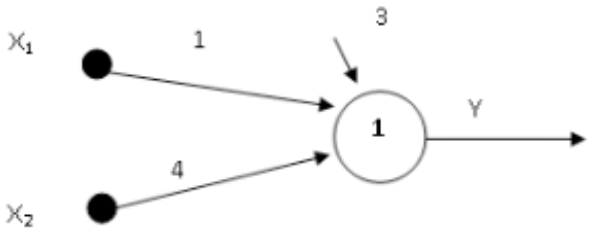
$W_{11}(novo) = 1 + 0*2 = 1$

$W_{21}(novo) = W_{21}(antigo) + X_2 Y_1$

$W_{21}(novo) = 2 + 1*2 = 4$

$b_1(novo) = b_1(antigo) + 1*Y_1$

$b_1(novo) = 2 + 1*1 = 3$



- Quarto par entrada-saída.

X1	X2	Y
0	0	-1

$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$

$v_1 = 0*1 + 0*4 + 1*3 = 3$

$$Y_1 = 1$$

$$W_{11}(\text{novo}) = W_{11}(\text{antigo}) + X_1 Y_1$$

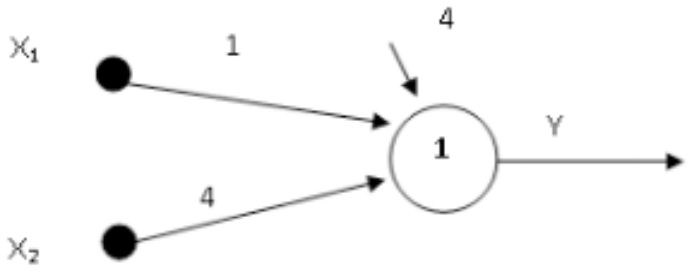
$$W_{11}(\text{novo}) = 1 + (0)*(1) = 1$$

$$W_{21}(\text{novo}) = W_{21}(\text{antigo}) + X_2 Y_1$$

$$W_{21}(\text{novo}) = 4 + (0)*(1) = 4$$

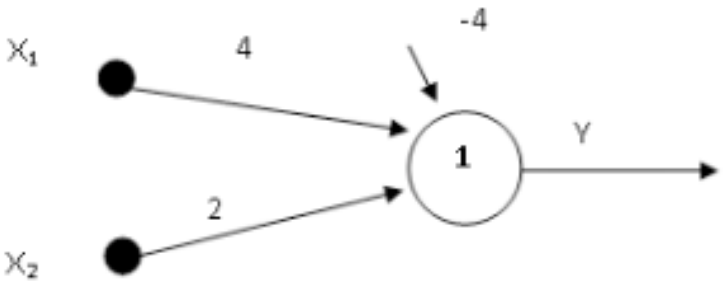
$$b_1(\text{novo}) = b_1(\text{antigo}) + 1*Y_1$$

$$b_1(\text{novo}) = 3 + 1*(1) = 4$$



Após várias apresentações do conjunto de treinamento os pesos não mais se alteram e a rede atinge estabilidade. Cada apresentação do conjunto de treinamento é chamada de época.

Esta é a configuração final da RNA e nela estão representados os pesos finais:



Após a apresentação de todos os pares Entrada/Saída, é possível testar a rede e verificar se ela aprendeu corretamente.

Conjunto de padrões da rede:

Entrada 1	Entrada 2	Saída
1	1	1
1	0	-1
0	1	-1
0	0	-1

- Quando se apresenta $X_1 = 1$ e $X_2 = 1$ a rede deve exibir na saída o valor 1.

$$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$$
$$v_1 = 1 * 4 + 1 * (2) + 1 * (-4) = 2 \Rightarrow Y_1 = 1$$

A rede acertou, produziu o valor de saída 1 como esperado.

- Para $X_1 = 1$ e $X_2 = 0$ a rede deve fornecer o valor de saída -1.

$$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$$
$$v_1 = 1 * 4 + 0 * (2) + 1 * (-4) = 0 \Rightarrow Y_1 = -1$$

A rede acertou, produziu o valor de saída -1 como esperado.

- Ao apresentar $X_1 = 0$ e $X_2 = 1$ a rede deve exibir o valor de saída -1.

$$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$$
$$v_1 = 0 * 4 + 1 * 2 + 1 * (-4) = -2 \Rightarrow Y_1 = -1$$

A rede acertou, produziu o valor de saída -1 como esperado.

- E finalmente, para $X_1 = 0$ e $X_2 = 0$ a rede deve apresentar na saída o valor -1.

$$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$$
$$v_1 = 0 * 4 + 0 * 2 + 1 * (-4) = -4 \Rightarrow Y_1 = -1$$

A rede acertou, produziu o valor de saída -1 como esperado.
Portanto, esta rede encontra-se devidamente treinada.
A seguir é apresentada uma tabela com os valores obtidos pela rede:

Época	b	x1	x2	Saída desejada	W0	W1	W2	Soma	Saída Obtida	Erro	Convergência
1	1	1	1	1	0,1	0,1	0,1	0,3	-1	2	
	1	1	0	1	0,4	0,1	0,1	0,3	-1	2	Não Convergiu
	1	0	1	1	0,6	0,4	0,1	0,4	-1	2	Não Convergiu
	1	0	0	-1	0,9	0,4	0,4	0,4	-1	0	Convergiu
2	1	1	1	1	0,9	0,4	0,4	0,4	-1	2	Não Convergiu
	1	1	0	1	1,1	0,6	0,6	0,4	-1	2	Não Convergiu
	1	0	1	1	1,4	0,9	0,6	0,4	-1	2	Não Convergiu
	1	0	0	-1	1,6	0,9	0,9	0,4	-1	0	Convergiu
3	1	1	1	1	1,6	0,9	0,9	0,4	-1	2	Não Convergiu
	1	1	0	1	1,9	1,1	1,1	0,4	-1	2	Não Convergiu
	1	0	1	1	2,1	1,4	1,1	0,4	-1	2	Não Convergiu
	1	0	0	-1	2,4	1,4	1,4	0,4	-1	0	Convergiu
4	1	1	1	1	2,4	1,4	1,4	0,4	-1	2	Não Convergiu
	1	1	0	1	2,6	1,6	1,6	0,4	-1	2	Não Convergiu
	1	0	1	1	2,9	1,9	1,6	0,4	-1	2	Não Convergiu
	1	0	0	-1	3,1	1,9	1,9	0,4	-1	0	Convergiu
5	1	1	1	1	3,1	1,9	1,9	0,4	-1	2	Não Convergiu
	1	1	0	1	3,4	2,1	2,1	0,4	-1	2	Não Convergiu
	1	0	1	1	3,6	2,4	2,1	0,4	-1	2	Não Convergiu
	1	0	0	-1	3,9	2,4	2,4	0,4	-1	0	Convergiu
6	1	1	1	1	3,9	2,4	2,4	0,4	-1	2	Não Convergiu
	1	1	0	1	4,1	2,6	2,6	0,4	-1	2	Não Convergiu
	1	0	1	1	4,4	2,9	2,6	0,4	-1	2	Não Convergiu
	1	0	0	-1	4,6	2,9	2,9	0,4	-1	0	Convergiu
7	1	1	1	1	4,6	2,9	2,9	0,4	-1	2	Não Convergiu
	1	1	0	1	4,9	3,1	3,1	0,4	-1	2	Não Convergiu
	1	0	1	1	5,1	3,4	3,1	0,4	-1	2	Não Convergiu
	1	0	0	-1	5,4	3,4	3,4	0,4	-1	0	Convergiu

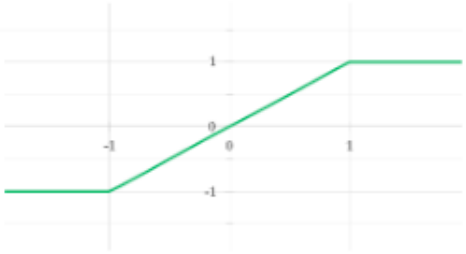
Exercícios Sugeridos

1) Considere o seguinte conjunto de treinamento, construa a rede neural com bias, inicialize os pesos e o bias 1 zero, utilize função de ativação limiar com θ igual a zero e treine a utilizando regra de Hebb, faça uma época.

X1	X2	Saída
1	1	1
1	-1	-1
-1	1	1
-1	-1	1

2) Repita o exercício (1) com os mesmos dados, mas com a seguinte função de ativação:

$$f(x) = \begin{cases} 1 \text{ se } x >= 1 \\ x \text{ se } -1 < x < 1 \\ -1 \text{ se } x <= -1 \end{cases}$$



3) Considere uma rede neural com um neurônio treine esta rede para simular o comportamento da função OR ou "OU" utilizando a regra de Hebb, função de ativação limiar com θ =1 , utilize duas épocas, inicialize os pesos e o bias com 0.1.

X1	X2	D
1	1	1
1	-1	1
-1	1	1
-1	-1	-1

4) Considere os parâmetros de uma rede neural. Construa a topologia da rede e calcule as saídas:

Matriz de pesos W= $\begin{pmatrix} 0.1 & 0.2 & 0.3 \\ 0.2 & 0.1 & 0.4 \end{pmatrix}$

$$f(v) = \begin{cases} 1 \text{ se } v >= 0 \\ -1 \text{ se } v < 0 \end{cases}$$

Vetor de bias b=[-0.5, 0.1, -0.4]

Vetor de entradas X=[1, 0.5]

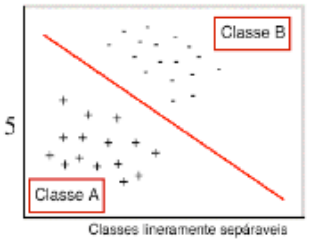
P) Redes Perceptron

Separabilidade Linear

Um problema é definido como linearmente separável se é possível representar o problema num plano bi-dimensional e traçar uma reta separando as classes do problema.

Exemplo: Considere a porta lógica *and*, cujos padrões são:

X1	X2	Y
1	1	1



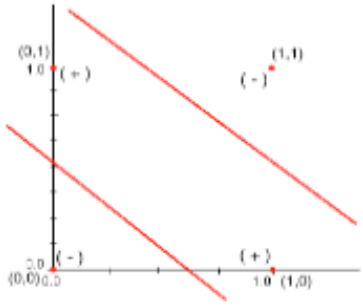
1	0	0
0	1	0
0	0	0

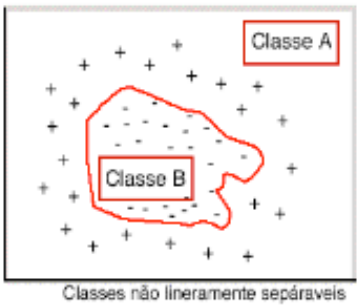
É possível separar através de uma reta as duas classes do problema. Portanto, todo problema linearmente separável pode ser resolvido por redes neurais de uma única camada e especialmente por perceptrons.

Considere o problema a seguir, que é não linearmente separável, como uma porta lógica XOR, “ou exclusivo”, cujos padrões são:

X1	X2	Y
1	1	0
1	0	1
0	1	1
0	0	0

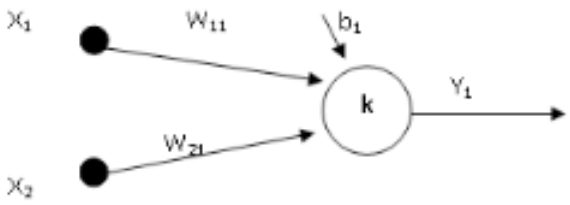
Neste caso, não é possível traçar apenas uma reta e que esta possa separar as diferentes classes do problema.





O tipo Perceptron

É um tipo de rede simples, de uma camada, com um ou mais neurônios de saída *j* conectados às entradas *i* através de pesos *Wij*. No caso mais simples, possui um só neurônio *k*.



O conjunto de equações da rede é dado por:

$$v = \sum_{i=0}^{n} w_{ij} x_i \qquad Y_i = F(v_i)$$

A função de ativação proposta por McCullock e Pitts, é uma função limiar:

$$f(\mathrm{v}) = \begin{cases} 1 \text{ se } \mathrm{v} >= 0 \\ 0 \text{ se } \mathrm{v} < 0 \end{cases}$$

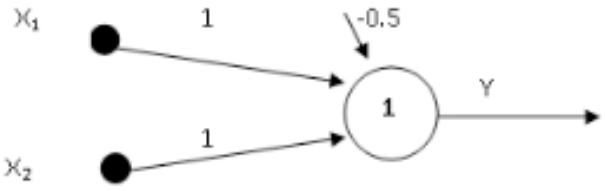
Para este caso, o limiar de ativação do neurônio, ou seja, o valor do qual o neurônio dispara, é zero, mas o modelo permite estabelecer um limiar de ativação f(x) diferente de zero. A função limiar se modifica.

$$f(\mathrm{v}) = \begin{cases} 1 \text{ se } \mathrm{v} >= 0 \\ 0 \text{ se } \mathrm{v} < 0 \end{cases}$$

Os neurônios da rede perceptron podem ser empregados para separar em classes distintas os padrões de entradas. Se a entrada líquida for maior que o limiar, o padrão dessa entrada pertence à classe 1, caso contrário, pertence à classe 0.

Como uma rede neural que simula o comportamento de uma porta lógica *and*.

X1	X2	Y	
1	1	1	Classe 1
1	0	0	Classe 0
0	1	0	
0	0	0	



A separação entre as duas classes é: $X_1w_{11} + X_2w_{21} + b_1 = 0$, ou seja, é uma equação pertencente a uma família de retas que podem separar as classes.

Algoritmo de treinamento do perceptron

Esta regra foi proposta por Rosemblatt e apresenta algumas mudanças em relação à regra de Hebb. A função de ativação passou a permitir um limiar Θ diferente de zero. Foi também introduzido um fator de aprendizado η (0 < η <= 1), que regula a velocidade de modificação dos pesos. O algoritmo de treinamento foi alterado, de forma que não seja feita a correção nos pesos quando a rede responde corretamente ao padrão.

O algoritmo completo é o seguinte, considerando X como o conjunto de entrada e D como o conjunto de saída, ambos valores reais e não obtidos pela rede.

Inicie aleatoriamente os pesos e bias: sugere-se iguais a 0;

1- Repita enquanto os pesos ainda estiverem sofrendo alterações

Para cada padrão (entrada e saída desejada) faça

Calcule as saídas.

se Y <> D então

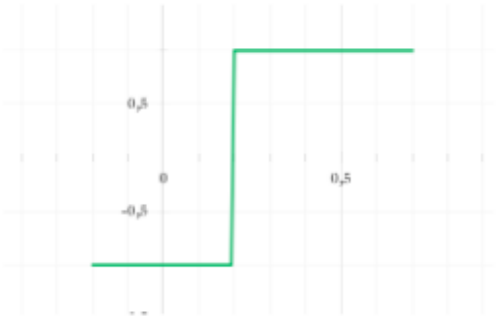
para cada sinapse faça

Wij(novo)= Wij(antigo)+ η * **Xi** * Dj

Exemplo:

Aplicar o algoritmo de aprendizado do Perceptron para treinar uma rede neural que simule o comportamento da função lógica *and* ou (e).

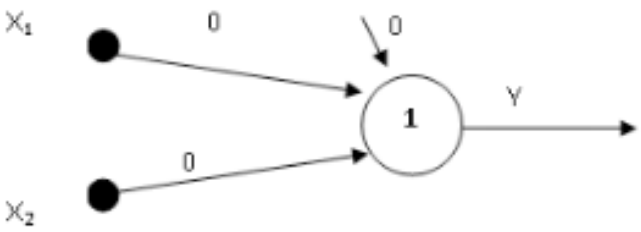
Utilizar-se-á a seguinte função de ativação, $\eta = 1$ e $\Theta = 0.2$.



$$f(v) = \begin{cases} 1 & \text{se } v >= 0.2 \\ -1 & \text{se } v < 0.2 \end{cases}$$

Entrada 1	Entrada 2	Saída
1	1	1
1	0	-1
0	1	-1
0	0	-1

Passo 1: Inicializar os pesos das sinapses com zero



Para cada par entrada saída, ajuste os pesos das sinapses.

- Primeiro par entrada-saída.

X1	X2	D
1	1	1

$$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$$

$$v_1 = 0*1 + 0*1 + 0*1 = 0$$

$Y_1 = -1$

$D_1 = 1$

$W_{11}(novo) = W_{11}\ (antigo) + \eta * X_1 * D_1$

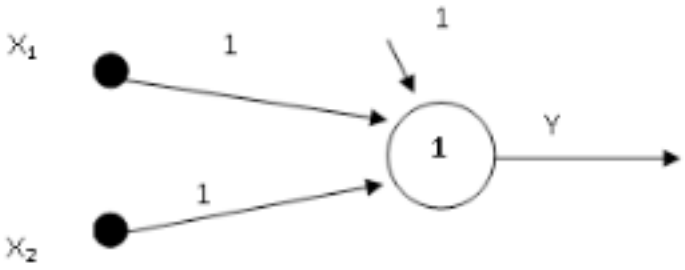
$W_{11}\ (novo) = 0 + 1*1*(1) = 1$

$W_{21}\ (novo) = W_{21}(antigo) + \eta * X_2 * D_1$

$W_{21}\ (novo) = 0 + 1*1*(1) = 1$

$b_1(novo) = b_1\ (antigo) + \eta * 1 * D_1$

$b_1(novo) = 0 + 1*1*(1) = 1$



- Segundo par entrada-saída.

X1	X2	D
1	0	-1

$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$

$v_1 = 1*1 + 1*0 + 1*1 = 2$

$Y_1 = 1$

$D_1 = -1$

$W_{11}\ (novo) = W_{11}\ (antigo) + \eta * X_1 * D_1$

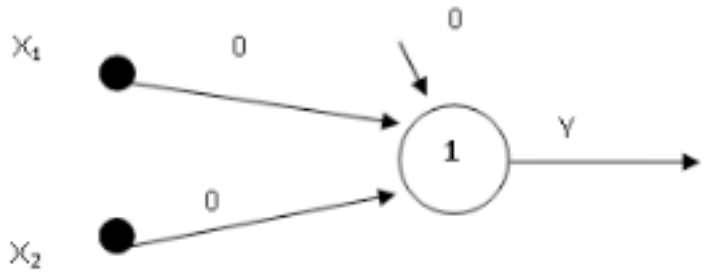
$W_{11}\ (novo) = 1 + 1*1*(-1) = 0$

$W_{21}\ (novo) = W_{21}\ (antigo) + \eta * X_2 * D_1$

$W_{21}\ (novo) = 1 + 1*0*(-1) = 0$

$b_1\ (novo) = b_1\ (antigo) + \eta * 1 * D_1$

$b_1\ (novo) = 1 + 1*1*(-1) = 0$



- Terceiro par entrada-saída.

X1	X2	D
0	1	-1

$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$

$v_1 = 0*0 + 1*0 + 1*0 = 0$

$Y_1 = -1$

$D_1 = -1$

$Y = D$, portanto não há alteração.

- Quarto par entrada-saída.

X1	X2	D
0	0	-1

$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$

$v_1 = 0*0 + 0*0 + 0*-1 = 0$

$Y_1 = -1$ e $D_1 = -1$

Y é igual a D e, neste caso, não há alteração. Segue-se para o próximo passo. Cada passada do conjunto de treinamento é chamada de época. Desta forma, mostrou-se a primeira época.

- Segunda época.

X1	X2	1	V	Y	D	W1	W2	b
1	1	1	-1	-1	1	0	0	-1
1	0	1	1	1	-1	0	1	-1
0	1	1	0	0	-1	0	0	-2
0	0	1	-2	-1	-1	0	0	-2

- Terceira época.

X1	X2	1	V	Y	D	W1	W2	b
1	1	1	-2	-1	1	1	1	-1
1	0	1	0	0	-1	0	1	-2
0	1	1	-1	-1	-1	0	1	-2
0	0	1	-2	-1	-1	0	1	-2

- Quarta época.

X1	X2	1	V	Y	D	W1	W2	b
1	1	1	-1	-1	1	1	2	-1
1	0	1	0	0	-1	0	2	-2
0	1	1	0	0	-1	0	1	-3
0	0	1	-3	-1	-1	0	1	-3

- Quinta época.

X1	X2	1	V	Y	D	W1	W2	b
1	1	1	-2	-1	1	1	2	-2
1	0	1	-1	-1	-1	1	2	-2
0	1	1	0	0	-1	1	1	-3
0	0	1	-3	-1	-1	1	1	-3

- Sexta época.

X1	X2	1	V	Y	D	W1	W2	b
1	1	1	-1	-1	1	2	2	-2
1	0	1	0	0	-1	1	2	-2
0	1	1	-1	-1	-1	1	2	-3
0	0	1	-3	-1	-1	1	2	-3

- Sétima época.

X1	X2	1	V	Y	D	W1	W2	b
1	1	1	0	0	1	2	3	-2
1	0	1	0	0	-1	1	3	-3
0	1	1	0	0	-1	1	2	-4
0	0	1	-4	-1	-1	1	2	-4

- Oitava época.

X1	X2	1	V	Y	D	W1	W2	b
1	1	1	-1	-1	1	2	3	-3
1	0	1	-1	-1	-1	2	3	-3
0	1	1	0	0	-1	2	2	-4
0	0	1	-4	-1	-1	2	2	-4

f. Nona época.

X1	X2	1	V	Y	D	W1	W2	b
1	1	1	0	0	1	3	3	-3
1	0	1	0	0	-1	2	3	-4
0	1	1	-1	-1	-1	2	3	-4
0	0	1	-4	-1	-1	2	3	-4

- Décima época.

X1	X2	1	V	Y	D	W1	W2	b
1	1	1	1	1	1	2	3	-4
1	0	1	-2	-1	-1	2	3	-4
0	1	1	-1	-1	-1	2	3	-4
0	0	1	-4	-1	-1	2	3	-4

Após 10 épocas a rede neural se estabilizou, ou seja, seus pesos não sofrem mais alterações. Agora pode-se testar e verificar se a rede aprendeu de fato a reconhecer os padrões ensinados.

Considere o seguinte conjunto de padrões da rede

Entrada 1	Entrada 2	Saída
1	1	1
1	0	-1
0	1	-1
0	0	-1

- Quando é apresentado o par X1 = 1 e X2 = 1 a rede deve tem na saída o valor 1.

$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$

$v_1 = 1 * 2 + 1 * 3 + 1 * (-4) = 1$

$Y_1 = 1$

A rede acertou, produziu o valor de saída 1 como esperado.

- Quando se apresenta o par $X_1 = 1$ e $X_2 = 0$ a rede deve ter como valor de saída -1.

$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$

$v_1 = 1 * 2 + 0 * 3 + 1 * (-4) = -2$

$Y_1 = -1$

A rede acertou, produziu o valor de saída -1 como esperado.

- Ao fornecer o par X1 = 0 e X2 = 1 a rede deve apresentar na saída o valor -1.

$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$

$v_1 = 0 * 2 + 1 * 3 + 1 * (-4) = -1$

$Y_1 = -1$

A rede acertou, produziu o valor de saída -1 como esperado.

- Ao apresentar o par $X_1 = 0$ e $X_2 = 0$ a rede deve ter como valor de saída -1.

$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$

$v_1 = 0 * 2 + 0 * 3 + 1 * (-4) = -4$

$Y_1 = -1$

A rede acertou, produziu o valor de saída -1 como esperado.

Exercícios Resolvidos

1) Considere o seguinte conjunto de treinamento:

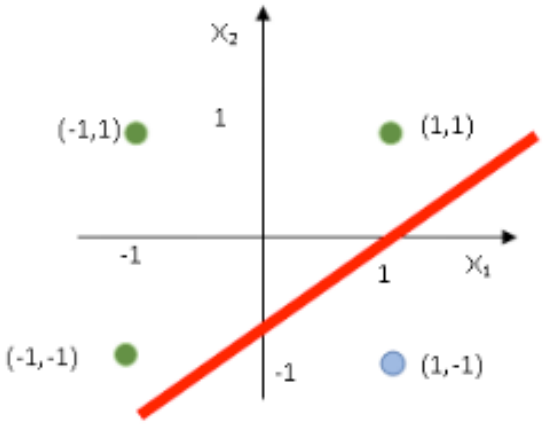
X1	X2	Saída
1	1	1
1	-1	-1
-1	1	1
-1	-1	1

a) Esboce o gráfico de X1 por X2 e diga se o problema é linearmente separável.

b) Quantas camadas e quantos neurônios deve ter a rede neural para reconhecer os dados?

Resolução:

a)



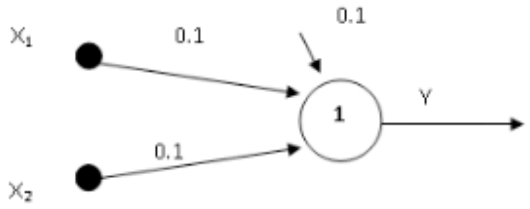
Sim, o problema é linearmente separável pois é possível separar as classes através de uma família de retas.

b) Uma camada e um neurônio, pois o problema é linearmente separável e possui saída binária.

2) Considere o conjunto de treinamento apresentado. Treine a rede utilizando o algoritmo do perceptron durante uma época, considere η = 0.5. Inicialize os pesos e o bias com 0.1, utilize função de ativação limiar com θ igual a zero.

X1	X2	Saída
1	1	1
1	-1	-1
-1	1	1
-1	-1	1

Resolução:



Para cada par entrada saída, ajuste os pesos das sinapses.

- Primeiro par entrada-saída.

X1	X2	D
1	1	1

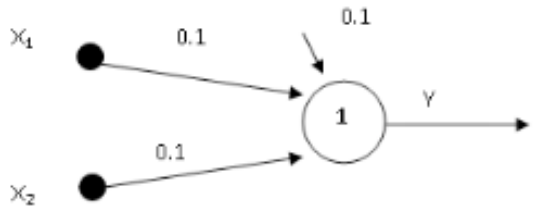
$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$

$v_1 = 0.1*1 + 0.1*1 + 0.1*1 = 0.3$

$Y_1 = 1$

$D_1 = 1$

Y_1 é igual a D_1 não há alterações.



- Segundo par entrada-saída.

X1	X2	D
1	-1	-1

$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$

$v_1 = 1*0.1 + (-1)*0.1 + 1*0.1 = 0.1$

$Y_1 = 1$

$D_1 = -1$

$W_{11}(novo) = W_{11}(antigo) + \eta * X_1 * D_1$

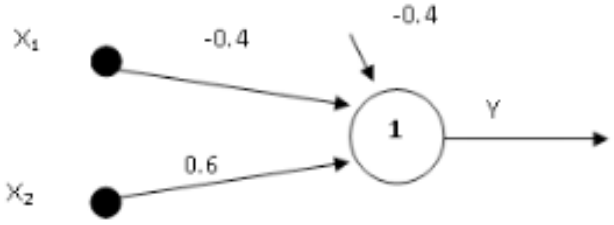
W_{11}(novo) = 0.1 + 0.5*1*(-1) = -0.4

W_{21}(novo) = W_{21} (antigo) + η * X_2* D_1

W_{21} (novo) = 0.1 + 0.5*(-1)*(-1) = 0.6

b_1(novo) = b_1(antigo) + η * **1*** d

b_1(novo) = 0.1 + 0.5*1*(-1) = - 0.4



- Terceiro par entrada-saída.

X1	X2	D
-1	1	-1

$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1*b_1$

v_1 = (-1)*(-0.4) + 1*0.6 + 1*(-0.4) = 0.6

$Y_1 = 1$ ® $D_1 = -1$

W_{11}(novo) = W_{11}(antigo) + η * X_1* D_1

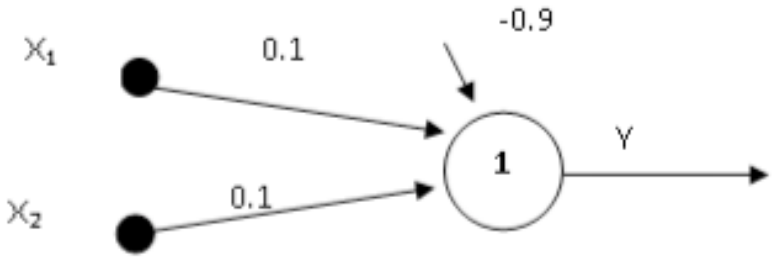
W_{11}(novo) = -0.4 + 0.5*(-1)*(-1) = 0.1

W_{21}(novo) = W_{21}(antigo) + η * X_2* D_1

W_{21}(novo) = 0.6 + 0.5*(1)*(-1) = 0.1

b_1(novo) = b_1(antigo) + η * 1* D_1

b_1(novo) = (-0.4) + 0.5*1*(-1)= - 0.9



- Quarto par entrada-saída.

X1	X2	D
-1	-1	1

$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$

$v_1 = (-1)*(0.1) + (-1)*0.1 + 1*(-0..9) = -1.1$

$Y_1 = -1$

$D_1 = 1$

$W_{11}(novo) = W_{11}(antigo) + \eta * X_1 * D_1$

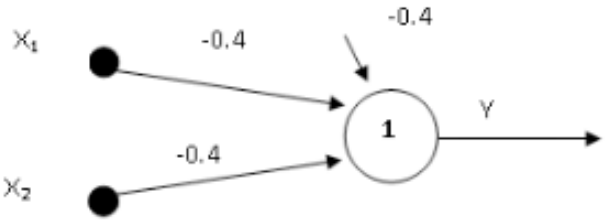
$W_{11}(novo) = 0.1 + 0.5*(-1)*(1) = -0.4$

$W_{21}(novo) = W_{21}(antigo) + \eta * X_2 * D_1$

$W_{21}(novo) = 0.1 + 0.5*(-1)*(1) = -0.4$

$b_1(novo) = b_1(antigo) + \eta * 1 * d_1$

$b_1(novo) = (-0.9) + 0.5*1*(1) = -0.4$



Os resultados, gerados em uma planilha, são apresentados em uma tabela:

Época	b	x1	x2	Saída desejada	W0	W1	W2	Soma	Saída Obtida	Erro	Convergência	Learning Rate
1	1	1	1	1	0,1	0,1	0,1	0,3	1	0		0,5
	1	1	-1	-1	0,1	0,1	0,1	0,4	1	-2	Não Convergiu	
	1	-1	1	-1	-0,4	-0,4	0,6	0,3	1	-2	Não Convergiu	
	1	-1	-1	1	-0,9	0,1	0,1	0,0	-1	2	Não Convergiu	
2	1	1	1	1	-0,4	-0,4	-0,4	0,1	-1	2	Não Convergiu	
	1	1	-1	-1	0,1	0,1	0,1	-0,2	-1	0	Convergiu	
	1	-1	1	-1	0,1	0,1	0,1	-0,2	-1	0	Convergiu	
	1	-1	-1	1	0,1	0,1	0,1	-0,2	-1	2	Não Convergiu	
3	1	1	1	1	0,6	-0,4	-0,4	-0,2	-1	2	Não Convergiu	
	1	1	-1	-1	1,1	0,1	0,1	-0,2	-1	0	Convergiu	
	1	-1	1	-1	1,1	0,1	0,1	-0,2	-1	0	Convergiu	
	1	-1	-1	1	1,1	0,1	0,1	-0,2	-1	2	Não Convergiu	
4	1	1	1	1	1,6	-0,4	-0,4	-0,2	-1	2	Não Convergiu	
	1	1	-1	-1	2,1	0,1	0,1	-0,2	-1	0	Convergiu	
	1	-1	1	-1	2,1	0,1	0,1	-0,2	-1	0	Convergiu	
	1	-1	-1	1	2,1	0,1	0,1	-0,2	-1	2	Não Convergiu	
5	1	1	1	1	2,6	-0,4	-0,4	-0,2	-1	2	Não Convergiu	
	1	1	-1	-1	3,1	0,1	0,1	-0,2	-1	0	Convergiu	
	1	-1	1	-1	3,1	0,1	0,1	-0,2	-1	0	Convergiu	
	1	-1	-1	1	3,1	0,1	0,1	-0,2	-1	2	Não Convergiu	
6	1	1	1	1	3,6	-0,4	-0,4	-0,2	-1	2	Não Convergiu	
	1	1	-1	-1	4,1	0,1	0,1	-0,2	-1	0	Convergiu	
	1	-1	1	-1	4,1	0,1	0,1	-0,2	-1	0	Convergiu	
	1	-1	-1	1	4,1	0,1	0,1	-0,2	-1	2	Não Convergiu	
7	1	1	1	1	4,6	-0,4	-0,4	-0,2	-1	2	Não Convergiu	
	1	1	-1	-1	5,1	0,1	0,1	-0,2	-1	0	Convergiu	
	1	-1	1	-1	5,1	0,1	0,1	-0,2	-1	0	Convergiu	
	1	-1	-1	1	5,1	0,1	0,1	-0,2	-1	2	Não Convergiu	

Dados de Entrada

X1	X2	Y	b
1	1	1	1
1	-1	-1	
-1	1	-1	
-1	-1	1	

Este é o bias

Função a ser treinada para obter. Neste caso, a função

Estes valores se repetem, pois representam os pares E/S do conjunto de treinamento

Uma época é um apresentação de um conjunto de treinamento completo. Neste caso, cada época possui 4 pares de Entrada/Saída (E/S).

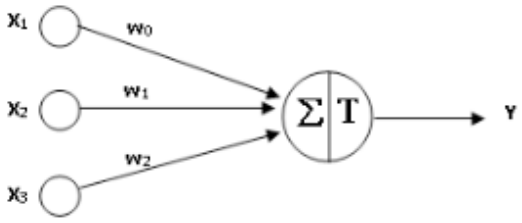
Os pesos são calculados (exceto para a primeira linha de dados, que foi definido como 0,0,0

Todos estes valores são calcuados, conforme a fórmula utilizada no exemplo.

3) Considere o conjunto de treinamento apresentado na tabela. Treine uma rede utilizando o algoritmo do perceptron de forma que a RNA seja capaz de reconhecer 2 frutas e 2 legumes.

		0	1
		Fruta	Legume
100	Maçã	X	
101	Banana	X	
110	Batata		X
111	Cenoura		X

Antes de iniciar o treinamento, codifica-se a informação e define-se as entradas e saídas desejadas. Para gerar um Perceptron, os valores devem assumir condições binárias, ou seja, 1 ou 0. Neste caso, uma proposta de identificação dos conjuntos da tabela já se encontra codificada. A saída se constitui de apenas duas posições, Fruta (0) ou Legume (1), logo, é possível usar um neurônio de saída que, identificando 100 (Maçã) ou 101 (Banana) produza 0 (Fruta), e, identificando 110 (Batata) ou 111 (Cenoura) produza 1 (Legume) como saída. Para as entradas usaremos 3 elementos: um para o viés (1) e os outros para as informações relativas aos ídolos (00, 01, 10 e 11).



Iniciando os pesos do neurônio com 0, tem-se o seguinte conjunto de pesos:

$$w = \{w_0, w_1, w_2\} = \{0, 0, 0\}$$

A função de transferência do neurônio é binária, portanto, se soma ponderada > 0, então saída= 1. Caso contrário, a saída= 0.

O conjunto de treinamento constitui os seguintes pares:

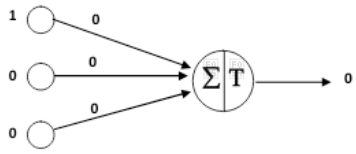
entrada 1= {x0, x1, x2}= {1, 0, 0}, cuja saída é saída 1= {y}= {0}

entrada 2= {x0, x1, x2}= {1, 0, 1}, cuja saída é saída 2= {y}= {0}

entrada 3= {x0, x1, x2}= {1, 1, 0}, cuja saída é saída 3= {y}= {1}

entrada 4= {x0, x1, x2}= {1, 1, 1}, cuja saída é saída 4= {y}= {1}

- 1ª vez da entrada 1 (Maçã):



Calculando a saída para a entrada 1

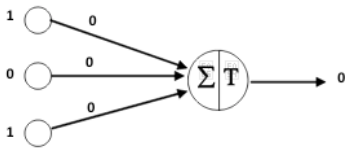
$\sum x.w = x0\ w0 + x1\ w1 + x2\ w2$

Soma = 1.0 + 0.0 + 0.0=0 e FT(0) = 0

saída = 0 (Fruta)

Saída desejada = 0 (Fruta), logo, a saída está correta!

- 1ª vez da entrada 2 (Banana):



Calculando a saída para a entrada 2

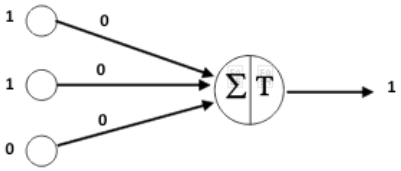
$\sum x.w = x0\ w0 + x1\ w1 + x2\ w2$

Soma = 1.0 + 0.0 + 1.0=0 e FT(0) = 0

saída = 0 (Fruta)

Saída desejada = 0 (Fruta), logo, a saída está correta!

- 1ª vez da entrada 3 (Batata):



Calculando a saída para a entrada 3

$\sum x.w = x0\ w0 + x1\ w1 + x2\ w2$

Soma = 1.0 + 1.0 + 0.0=0 e FT(0) = 0

saída = 0 (Fruta)

Saída desejada = 1 (Legume): não está correta!

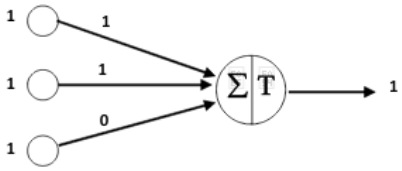
Como saída = 0: adicionar a cada peso a sua entrada:

w0 = 0 + 1 = 1

w1 = 0 + 1 = 1

w2 = 0 + 0 = 0

- 1ª vez da entrada 4 (Cenoura):



Calculando a saída para a entrada 4
$\sum x.w = x0\ w0 + x1\ w1 + x2\ w2$
Soma = 1.1 + 1.1 + 1.0=0 e FT(2) = 1
saída = 1 (Legume)
Saída desejada = 1 (Legume): saída está correta!

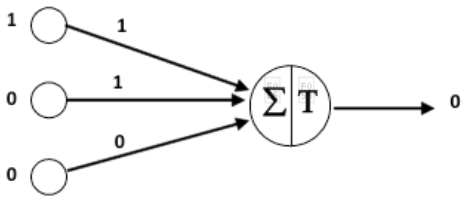
Agora que todas as entradas do conjunto de treinamento foram apresentadas, deve-se avaliar o treinamento. Essa avaliação ajuda a decidir se a rede está ou não devidamente treinada (reconhece ou não todos os conjuntos).

Uma forma de avaliação é fazer um teste total, ou seja, apresentar todo o conjunto de treinamento e quando uma entrada não for reconhecida, ainda durante a apresentação do conjunto, interromper o teste, ajustar os pesos e continuar o treinamento. Outra proposta é realizar um teste por amostragem, quando o conjunto for muito grande. Quer dizer, fazer um teste com 10%, ou 20%, conforme preferência do projetista da rede, do conjunto de treinamento e verificar o seu sucesso.

Outro ponto importante é a precisão da rede. Dependendo do conjunto de treinamento e da arquitetura da RNA, talvez ela não reconheça 100% dos casos apresentados, mas reconhece uma percentagem razoável. Essa decisão deve ser tomada em tempo de treinamento. Para este caso em particular, será decidido pelo treinamento total, ou seja, 100% de reconhecimento, já que o conjunto e a rede são pequenos e simples.

O treinamento será reiniciado, já que aconteceu um reajuste de peso no treinamento anterior, ou seja, alguma entrada não foi reconhecida apropriadamente, e, dessa forma, todo conjunto de treinamento será repassado, para eliminar a possibilidade da rede não reconhecer alguma entrada do conjunto apresentado.

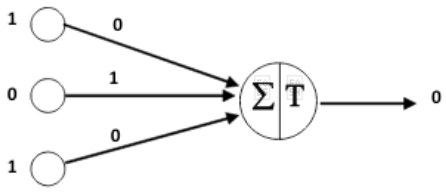
- 2ª vez da entrada 1 (Maçã):



Calculando a saída para a entrada 1
$\sum x.w = x0\ w0 + x1\ w1 + x2\ w2$
Soma = 1.1 + 0.1 + 0.0=0 e FT(1) = 1
saída = 1 (Legume) e
Saída desejada = 0 (Fruta), logo, a saída incorreta!

Como saída = 1, deve-se subtrair de cada peso a sua entrada:
w0 = 1 – 1 = 0 w1 = 1 – 0 = 1 w2 = 0 – 0 = 0

- 2ª vez a entrada 2 (Banana):



Calculando a saída para a entrada 2

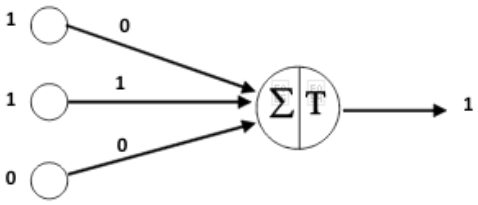
$\sum x.w = x0\ w0 + x1\ w1 + x2\ w2$

Soma = 1.0 + 0.1 + 1.0=0 e FT(0) = 0

saída = 0 (Fruta)

Saída desejada = 0 (Fruta), logo, a saída está correta!

- 2ª vez a entrada 3 (Batata):



Calculando a saída para a entrada 3

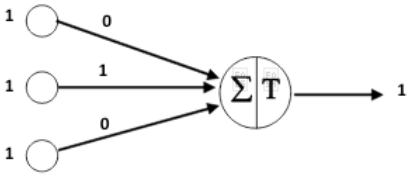
$\sum x.w = x0\ w0 + x1\ w1 + x2\ w2$

Soma = 1.0 + 1.1 + 0.0=0 e FT(1) = 1

saída = 1 (Legume)

Saída desejada = 1 (Legume):a saída está correta!

- 2ª vez a entrada 4 (Cenoura):



Calculando a saída para a entrada 4

$\sum x.w = x0\ w0 + x1\ w1 + x2\ w2$

Soma = 1.0 + 1.1 + 1.0=0 e FT(1) = 1

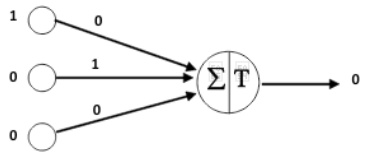
saída = 1 (Legume)

Saída desejada = 1 (Legume): a saída está correta!

No segundo treinamento, aconteceu um ajuste de pesos, portanto reinicia-se todo o treinamento da rede neural. Entretanto, pode-se dispensar todo este retreino, pois as entradas 2, 3 e 4 produzem saídas corretas mediante os pesos { 0, 1, 0 } da rede.

Deve-se apenas verificar se a saída produzida para a entrada 1 está correta, pois não necessariamente ajustando os pesos uma vez fará com que se produza a saída esperada.

- Reapresentando a entrada 1 (Maçã):



Calculando a saída para a entrada 1
$\sum x.w = x0\ w0 + x1\ w1 + x2\ w2$
Soma = 1.1 + 0.1 + 0.0=0 e FT(0) = 0
saída = 0 (Fruta)
Saída desejada = 0 (Fruta), logo, a saída está correta!

Como foi reconhecida a Fruta Maçã, a rede reconheceu todos os casos apresentados do conjunto de treinamento. Portanto, rede devidamente treinada.

Exercícios Propostos

1) Considere uma rede neural com um neurônio treine esta rede para simular o comportamento da função OR ou "OU" algoritmo do perceptron, função de ativação limiar com θ =1 , utilize duas épocas, inicialize os pesos com 0.2. Utilize η=0.5

X1	X2	D
1	1	1
1	-1	1
-1	1	1
-1	-1	-1

2) Considere a seguinte rede neural de uma camada, com duas entradas, três neurônios na camada de saída, função de ativação limiar com θ = 0.2 e com os seguintes parâmetros:

Matriz de pesos W= $\begin{pmatrix} 0.3 & 0.4 & 0.5 \\ 0.6 & 0.2 & 0.7 \end{pmatrix}$

Vetor de bias b=[-0.3, -0.5, 0.8]

Vetor de entradas X=[1, 0.4]

Vetor de saídas D=[1, 1 , -1]

Construa a topologia da rede e calcule as saídas Y1, Y2 e Y3.

Treine a rede durante uma época com o algoritmo do perceptron. Use η = 0.5

3) Sabe–se que os vetores (1, 1, 1, 1) e (–1, 1, –1, –1) são membros de uma classe, enquanto os vetores (1, 1, 1, –1) e (1, –1, –1, 1) não são. Apresente o treinamento de uma rede de Perceptron para classificar os vetores e apresente os resultados obtidos. Utilize função de ativação limiar com θ = 1, Use η = 0.5, treine por duas épocas. Diga se é necessário continuar o treinamento.

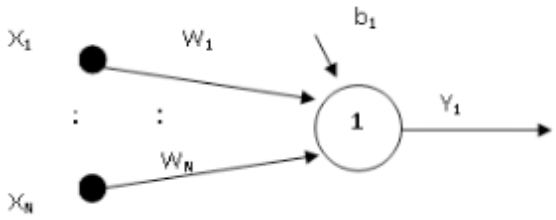
4) Seja o conjunto de treinamento abaixo. Treine uma rede neural com o algoritmo do perceptron, função de ativação limiar com θ =1 , utilize duas épocas, inicialize os pesos com 0.5. Utilize η=0.7. Diga se o problema é linearmente separável.

X1	X2	D
1	1	-1
1	-1	-1
-1	1	-1
-1	-1	1

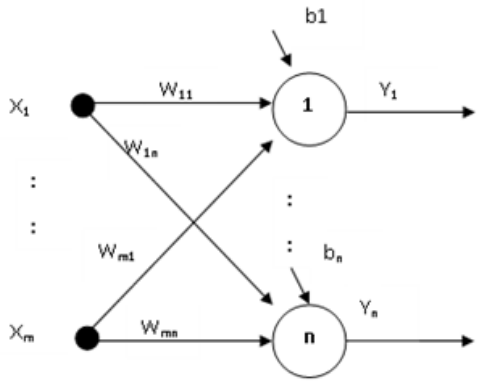
Q) Redes Adaline

O modelo de RNA Adaline, Widrow e Hoff em 1960, inicialmente foi chamado de (ADAptative LINear Element) e posteriormente de (ADAptative LInear NEuron). São RNA bem simples e possuem a arquitetura do neurônio idêntica a do Perceptron, tendo como diferencial a regra de aprendizado, neste caso chamada de regra delta.

Na sua forma mais simples é formada por apenas um neurônio, com varias entradas uma saída e um “bias”.



Da mesma forma que no Perceptron, a rede pode ter vários neurônios recebendo as mesmas entradas em uma única camada de saída.



A função de ativação utilizada pelas redes Adaline é a mesma utilizada pelo perceptron, ou seja, uma função bipolar (1, -1). Podendo ser a função limiar que se conhece com Θ ajustável.

$$f(v) = \begin{cases} \text{se } v >= \theta \\ -1 \text{ se } v < \theta \end{cases}$$

Pode-se utilizar ainda a seguinte função de ativação:

$$f(v) = \begin{cases} 1 \text{ se } v >= \theta \\ v \text{ se } -\theta < v < \theta \end{cases}$$

Algoritmo de Treinamento

A regra de aprendizado das redes Adaline é conhecida por regra delta, o princípio básico é ajustar os pesos a partir do erro na saída da rede. O erro, ou custo, para um neurônio é dado pela diferença entre a D_k é a saída desejada (fornecida no padrão *p* de entrada/saída) e Y_k é a saída realmente obtida (fornecida pela rede quando é apresentado o padrão *p*). O padrão p é qualquer membro do conjunto de treinamento. A regra é capaz de atualizar corretamente os pesos a cada iteração, com entradas binárias ou contínuas.

A tendência do erro a cada apresentação do conjunto de treinamento é diminuir até chegar bem próximo de zero. Quando se utiliza uma rede com mais de um neurônio na primeira camada, faz-se necessário calcular o erro na saída da rede e minimizá-lo. Portanto, se apenas somar os erros de cada neurônio i os erros negativos poderiam anular os erros positivos e dar uma falsa estabilização da rede. Para evitar este problema utiliza-se o erro quadrático, a função de erro, conforme equação (XVIII):

$$E = \frac{1}{2N} \sum_{i=1}^{N} [d_i - y_i]^2 \qquad \text{(XVIII)}$$

Onde *N* é o número total de neurônios na camada de saída. O fator ½ é unicamente para ajuste do algoritmo de treinamento. Como o erro *E* é função da saída da rede Y_k, e esta é função dos pesos *wi*, podemos dizer que o erro *E* é função dos pesos da rede. Ou seja: *E = f(w1, w2, ... ,wN)*.

Assim, a ideia é minimizar o erro ajustando os pesos, ou seja, encontrar o mínimo da função de erro. Este mínimo é dado pela derivada parcial da função de erro em relação a cada peso. O gradiente da função de erro é o vetor composto pelas derivadas parciais da função de erro em relação a cada peso e aponta para a direção de crescimento da função, da mesma forma que a derivada em uma função monovalorada *f(x)*. Assim, a ideia é variar os pesos de forma a caminhar na direção contrária ao gradiente, que é a direção de maior diminuição do valor da função de erro,

como na equação (XIX).

$$\Delta W_i = -\eta \frac{\partial E}{\partial W_i} \qquad \text{(XIX)}$$

Onde η é uma constante de proporcionalidade, também chamada de taxa de treinamento.

Realizando as devidas manipulações matemáticas, obtém-se a equação (XX).:

$$v_i = \sum_{i=0}^{N} x_i w_i = Y_i \qquad \Delta W_i = -\eta X_i(d_i - y_i)$$

$$Wij\ (novo) = Wij(antigo) + \eta * Xi * (Dj - Yj) \qquad \text{(XX)}$$

O Algoritmo de treinamento pode ser assim resumido:

1 - Inicie os pesos e bias com valores aleatórios pequenos

2 - Repita

Para cada padrão (entrada e saída desejada) faça

Calcule as saídas Y

Para cada conexão faça

Wij(novo) = Wij(antigo) + η * Xi * (Dj - Yj)

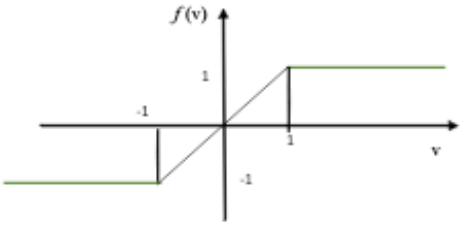
b(novo) = b(antigo) + η * 1* (Dj - Yj)

até que um erro pequeno tenha sido atingido

Aplicação

Construir uma RNA que simule o comportamento de uma porta lógica “OR”.

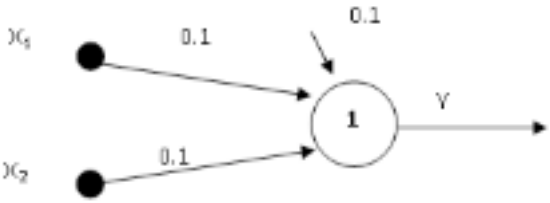
X1	X2	D
1	1	1
1	0	1
0	1	1
0	0	-1



Será utilizada a seguinte função de ativação.

$$f(v) = \begin{cases} 1 \text{ se } v >= 1 \\ v \text{ se } -1 < v < 1 \\ -1 \text{ se } v <= -1 \end{cases}$$

Passo 1 – Iniciar os pesos e o bias com valores aleatórios pequenos
Os pesos serão inicializados com 0.1 inclusive o bias. Será utilizado η =0.25.



- Primeiro par entrada-saída.

X1	X2	D
1	1	1

$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$

$v_1 = 1*0.1 + 1*0.1 + 1*0.1 = 0.3$

$Y_1 = 0.3$

$D_1 = 1$

$e_1 = D_1 - Y_1 = 1 - 0.3 = 0.7$

Como está entre -1 e 1, o valor de saída se mantém, conforme a função de ativação: $Y_{obtido} = 0,7$

$$W_{11}(novo) = W_{11}(antigo) + \eta * X_{11} * e_1$$

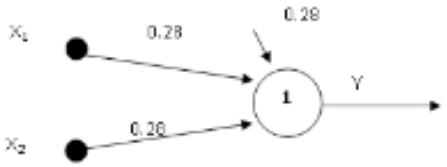
$$W_{11}(novo) = 0.1 + 0.25*1*0.7 = 0.28$$

$$W_{21}(novo) = W_{21}(antigo) + \eta * X_2 * e_1$$

$$W_{21}(novo) = 0.1 + 0.25*1*0.7 = 0.28$$

$$b_1(novo) = b_1(antigo) + \eta * 1 * e_1$$

$$b_1(novo) = 0.1 + 0.25*1*0.7 = 0.28$$



- Segundo par entrada-saída.

X1	X2	D
1	0	1

$$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1*b_1$$

$$v_1 = 1*0.28 + 0*0.28 + 1*0.28 = 0.56$$

$$Y_1 = 0.56$$

$$D_1 = 1$$

$$e_1 = D_1 - Y_1 = 1 - 0.56 = 0.44$$

$$W_{11}(novo) = W_{11}(antigo) + \eta * X_1 * e_1$$

$$W_{11}(novo) = 0.28 + 0.25*1*0.56 = 0.39$$

$$W_{21}(novo) = W_{21}(antigo) + \eta * X_2 * e_1$$

$$W_{21}(novo) = 0.28 + 0.25*0*0.56 = 0.28$$

$$b_1(novo) = b_1(antigo) + \eta * 1 * e_1$$

$$b_1(novo) = 0.28 + 0.25*1*0.56 = 0.39$$

- Terceiro par entrada-saída.

X1	X2	D
0	1	1

$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$

$v_1 = 0*0.41 + 1*0.28 + 1*0.41 = 0.66$

$Y_1 = 0.66$

$D_1 = 1$

$e_1 = D_1 - Y_1 = 1 - 0.66 = 0.34$

$W_{11}(novo) = W_{11}(antigo) + \eta * X_1 * e_1$

$W_{11}(novo) = 0.39 + 0.25*0*0.34 = 0.39$

$W_{21}(novo) = W_{21}(antigo) + \eta * X_2 * e_1$

$W_{21}(novo) = 0.28 + 0.25*1*0.34 = 0.36$

$b_1(novo) = b_1(antigo) + \eta * 1 * e_1$

$b_1(novo) = 0.39 + 0.25*1*0.34 = 0.47$

- Quarto par entrada-saída.

X1	X2	D
0	0	-1

$v_1 = X_1 * w_{11} + X_2 * w_{21} + 1 * b_1$

$v_1 = 0*0.41 + 0*0.36 + 1*0.49 = 0.47$

$Y_1 = 0.47$

$D_1 = 1$

$e_1 = D_1 - Y_1 = -1 - 0.49 = -1.49$

$W_{11}(novo) = W_{11}(antigo) + \eta * X_1 * e_1$

$W_{11}(novo) = 0.39 + 0.25*0*(-1.49) = 0.39$

$W_{21}(novo) = W_{21}(antigo) + \eta * X_2 * e_1$

$W_{21}(novo) = 0.36 + 0.25*0*(-1.49) = 0.36$

$b_1(novo) = b_1(antigo) + \eta * 1 * e$

$b_1(novo) = 0.47 + 0.25*1*0.39 = 0.47$

Uma planilha com os resultados das primeiras épocas está aqui apresentada:

Época	b	x1	x2	Saída desejada	W0	W1	W2	Soma	Saída Obtida	Erro	Convergência	Learning Rate
1	1	1	1	1	0,10	0,10	0,10	0,30	0,3	0,7		0,25
	1	1	0	1	0,28	0,28	0,28	0,55	0,55	0,45	Não Convergiu	
	1	0	1	1	0,39	0,39	0,28	0,66	0,66	0,34	Não Convergiu	
	1	0	0	-1	0,47	0,39	0,36	0,47	0,47	-1,47	Não Convergiu	
2	1	1	1	1	0,10	0,39	0,36	0,85	0,85	0,15	Não Convergiu	
	1	1	0	1	0,14	0,42	0,40	0,57	0,57	0,43	Não Convergiu	
	1	0	1	1	0,25	0,53	0,40	0,65	0,65	0,35	Não Convergiu	
	1	0	0	-1	0,34	0,53	0,49	0,34	0,34	-1,34	Não Convergiu	
3	1	1	1	1	0,00	0,53	0,49	1,02	1,00	0	Convergiu	
	1	1	0	1	0,00	0,53	0,49	0,54	0,54	0,46	Não Convergiu	
	1	0	1	1	0,12	0,65	0,49	0,60	0,60	0,4	Não Convergiu	
	1	0	0	-1	0,22	0,65	0,58	0,22	0,22	-1,22	Não Convergiu	
4	1	1	1	1	-0,09	0,65	0,58	1,15	1,00	0	Convergiu	
	1	1	0	1	-0,09	0,65	0,58	0,56	0,56	0,44	Não Convergiu	
	1	0	1	1	0,02	0,76	0,58	0,61	0,61	0,39	Não Convergiu	
	1	0	0	-1	0,12	0,76	0,68	0,12	0,12	-1,12	Não Convergiu	
5	1	1	1	1	-0,16	0,76	0,68	1,28	1,00	0	Convergiu	
	1	1	0	1	-0,16	0,76	0,68	0,60	0,60	0,4	Não Convergiu	
	1	0	1	1	-0,06	0,86	0,68	0,62	0,62	0,38	Não Convergiu	
	1	0	0	-1	0,04	0,86	0,78	0,04	0,04	-1,04	Não Convergiu	
6	1	1	1	1	-0,22	0,86	0,78	1,41	1,00	0	Convergiu	
	1	1	0	1	-0,22	0,86	0,78	0,64	0,64	0,36	Não Convergiu	
	1	0	1	1	-0,13	0,95	0,78	0,64	0,64	0,36	Não Convergiu	
	1	0	0	-1	-0,04	0,95	0,87	-0,04	-0,04	-0,96	Não Convergiu	
7	1	1	1	1	-0,28	0,95	0,87	1,53	1,00	0	Convergiu	
	1	1	0	1	-0,28	0,95	0,87	0,67	0,67	0,33	Não Convergiu	
	1	0	1	1	-0,20	1,03	0,87	0,67	0,67	0,33	Não Convergiu	
	1	0	0	-1	-0,12	1,03	0,95	-0,12	-0,12	-0,88	Não Convergiu	

Exercícios Propostos

1) Considere uma rede do tipo Adaline. Treine-a para verificar se uma pessoa está doente ou não, para isso considere as seguintes entradas:

Nome	Febre	Enjôo	Dor	Doente?
João	S	S	N	S
Maria	N	S	N	N
Pedro	N	S	S	S
Carlos	S	N	S	S
Joana	N	N	N	N

Considere os seguintes parâmetros

–η=1;

-Pesos iniciais 0.2

- Épocas = 2

- Após as duas épocas verifique se é necessário parar o treinamento.

- Faça um gráfico da evolução do erro durante o treinamento.

2) Considere os seguintes parâmetros da RNA, treine duas épocas, utilizando a regra delta.

Matriz de pesos W= $\begin{pmatrix} 0.3 & 0.4 & 0.5 \\ 0.4 & -0.3 & 0.6 \end{pmatrix}$

- Vetor de bias b=[0.5, 0.1, 0.4]

- Vetor de entradas X=[1, 0.5]
- Vetor de saídas D = [1, 1, 1]
– η=0.3
- Função de ativação $f(v) = \begin{cases} 1 \text{ se } v >= 0.4 \\ -1 \text{ se } v < 0.4 \end{cases}$

3) Considere a seguinte rede neural de uma camada, com três entradas, dois neurônios na camada de saída, função de ativação limiar com θ = 0.5 e seja os seguintes parâmetros:

- Matriz de pesos W= $\begin{pmatrix} 0.5 & 0.2 \\ 0.6 & 0.1 \\ 0.7 & 0.5 \end{pmatrix}$
- Vetor de bias b=[-0.2, 0.1]
- Vetor de entradas X=[1, 0.5, 0.3]
- Vetor de saídas D=[1, -1]
– η=0.7
- Treine duas épocas, utilizando a regra delta.

R) *Backpropagation*

Este algoritmo treina redes do tipo *feedforward* com neurônios com qualquer função de ativação que seja derivável.

Algoritmo Backpropagation

Este é um algoritmo de treinamento supervisionado para redes do tipo *feedforward* que tenham neurônios com qualquer função de ativação que seja derivável. Este algoritmo permite modificar aos poucos os valores das sinapses de modo a otimizar a saída da rede. para a atualização dos pesos de uma rede neural que já estão pré-definidos: as funções de ativação, os pesos, o passo de treinamento e o momento.

1º Passo Calcular, no sentido *feedforward*, o valor que entra em cada um dos neurônios, utilizando a fórmula (XXI):

$$\Delta W_i = - \eta \frac{\partial E}{\partial W_i} \qquad \text{(XXI)}$$

onde:

$u_i^c \rightarrow$ a entrada que chega no neurônio i da camada c;

$w_{ij} \rightarrow$ peso sináptico que liga o neurônio i (camada anterior) ao neurônio j (camada posterior);

$x_j \rightarrow$ neurônio i (camada anterior);

$b_j \rightarrow$ bias que está interligado ao neurônio j.

2º Passo Calcular, no sentido *feedforward*, o valor que sai de cada neurônio. A saída depende da função de ativação, como mostrado a seguir:

Linear → $v_j = u_j$;

Tangente hiperbólica ® $v_j = tgh\ u_j \approx u_j$;

Degrau → $v_j = 1$, se $u_j > 0$ e $v_j = 0$, se $u_j <= 0$.

3º Passo Os passos 1 e 2 devem ser repetidos para as demais camadas ocultas (caso existam) e para a camada de saída.

4º Passo Após calcular os valores da camada de saída, calcula-se o erro associado a todos os neurônios desta camada, utilizando a fórmula: $\varepsilon_i = y_{obtido} - y_{real}$

onde ε_i ® erro associado à diferença entre a saída obtida pela rede e a saída real.

5º Passo Calcular o ganho para pequenos sinais de saída g_i dos neurônios, da seguinte maneira:

Para neurônio tipo linear e degrau ® $g_i = 1$

Para neurônio tipo tgh ® $g_i = 1 - v_i^2$.

6º Passo Calcular o erro que será retropropagado, levando em consideração o ganho dos neurônios, usando a fórmula: $\delta_i = g_i * \varepsilon_i$

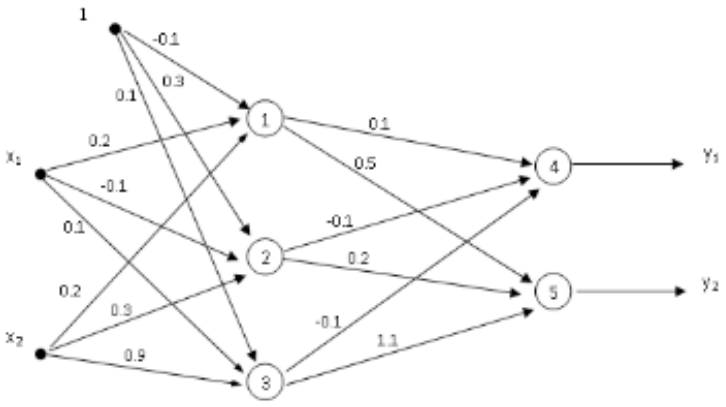
7º Passo Retropropagar o erro, utilizando o passo de treinamento α, a fim de calcular a variação das sinapses: $\Delta w_{ij} = 2\ \alpha\ v_i\ \delta_i$

8º Passo Atualizar as sinapses: $w_i^{novo} = w_i^{anterior} + \Delta w_{ij}$

Exercício Resolvido

Na rede neural apresentada a seguir, os neurônios 1, 2, 3 e 5 são do tipo tangente hiperbólica e o neurônio 4 é do tipo linear. O treinamento é do tipo Regra Delta com passo $\alpha = 0{,}1$. É apresentado o par x {0.1, 0.7} e y = { 0.2, 1}.

Quais os novos valores das sinapses após o primeiro passo de treinamento?



$$u_i^c = \sum_{j=1}^{n} w_{ij}^c * x_j + b_j$$

- 1ª camada

$u_1 = w_{11}\, x_1 + w_{12}\, x_2 + b_1 = 0.2 * 0.1 + 0.2 * 0.7 + 1 * (-0.1) = 0.06$

$u_2 = 0.1 * (-0.1) + 0.7 * 0.3 + 1 * 0.3 = 0.5$

$u_3 = 0.1 * 0.1 + 0.7 * 0.9 + 0.1 * 1 = 0.74$

$v_j = \text{tgh}\, u_j \approx u_j$

$v_1 = 0.06$

$v_2 = 0.5$

$v_3 = 0.74$

- 2ª camada

$u_4 = v_1\, w_{41} + v_2 + w_{42} = 0.06 * 0.1 + 0.5 * (-0.1) + 0.74 * (-0.1) = -0.118$

$u_5 = 0.06 * 0.5 + 0.5 * 0.2 + 0.74 * 1.1 = 0.94$

$v_j = \text{tgh}\, u_j \approx u_j$ ® $v_4 = -0.118$

$v_j = u_j$ ® $v_5 = 0.94$

- Cálculo do Erro: $\varepsilon_i = y_{obtido} - y_{real}$

$\varepsilon_1 = 0.2 - (-0.078) = 0.318$

$\varepsilon_2 = 1.0 - 0.93 = 1{,}94$

- Cálculo do ganho

$$\begin{cases} \text{neurônio linear} \rightarrow g_i = 1 \\ \text{neurônio tgh} \rightarrow g_i = 1 - v_i^2 \end{cases}$$

$g_1 = 1 - (0.06)^2 = 0.996$

$g_2 = 1-(0.5)^2 = 0.75$

$g_3 = 1-(0.74)^2 = 0.452$

$g_4 = 1$ (linear)

$g_5 = 1-(0.93)^2 = 0.116$

Calculo do δ $\qquad \delta i = gi * \varepsilon i$

$\delta_4 = g_4 * \varepsilon_4 = 1 * 0.318 = 0.318$

$\delta_5 = g_5 * \varepsilon_2 = 0.116 * 1{,}94 = 0.225$

$\delta_1 = g_1 * (w_{14} * \delta_4 + w_{15} * \delta_5) = 0.996 * (0.1 * 0.318 + 0.5 * 0.225) = 0.143$

$\delta_2 = g_2 * (w_{24} * \delta_4 + w_{25} * \delta_5) = 0.75 * (-0.1 * 0.318 + 0.2 * 0.225) = 0.010$

$\delta_3 = g_3 * (w_{34} * \delta_4 + w_{35} * \delta_5) = 0.452 * (-0.1 * 0.318 + 1.1 * 0.225) = -0.097$

Calculo da variação do peso $\Delta wij = 2\ \alpha\ vi\ \delta i$

$\Delta w_{41} = 2\ \alpha\ v_4\ \delta_4 = 2 * 0.1 * (-0,118) * 0.318 = -0.0075$

$\Delta w_{51} = 2\ \alpha\ v_5\ \delta_5 = 2 * 0.1 * (0,94) * 0.225 = 0.0423$

$\Delta w_{3\,x1} = 2\ \alpha\ x_2\ \delta_3 = 2 * 0.1 * 0,7 * (0.097) = 0.0136$

Atualização dos pesos $w_i^{novo} = w_i^{anterior} + \Delta w_{ij}$

$w_{41} = 0.1 + (-.0075) = 0.0925$

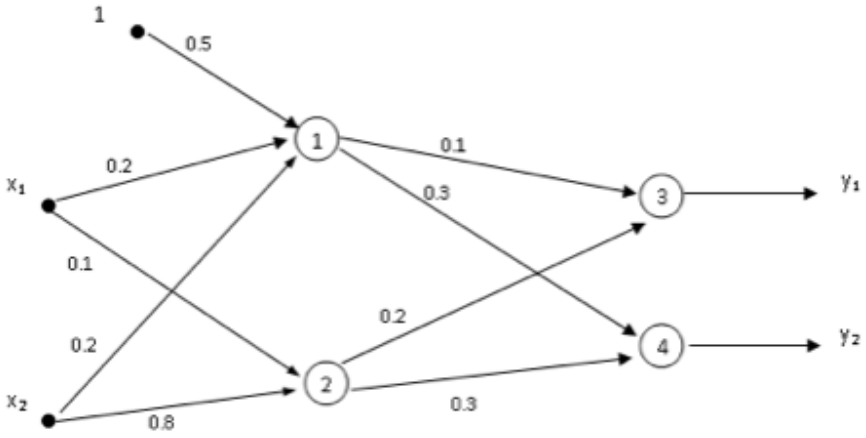
$w_{51} = 0.5 + (0.0423) = 0.5423$

$w_{3\,x1} = 0.1 + 0.0136 = 0.1136$

Exercício Proposto:

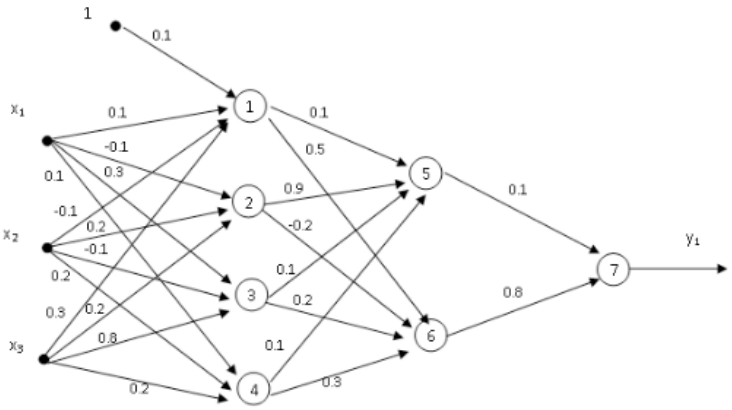
1) Considere esta rede apresentada. Os neurônios 1, 2, 3 e 4 são do tipo tangente hiperbólica. O treinamento é do tipo Regra Delta com passo $\alpha = 0,13$. É apresentado o par x {0.2, 0.5} e y = {0.3, 0.8}.

Quais os novos valores das sinapses após o primeiro passo de treinamento?



2) Nesta RNA, os neurônios 1, 2, 4 e 6 são do tipo tangente hiperbólica, os neurônios 3 e 5 são do tipo linear e o neurônio 7 é do tipo degrau. O treinamento é do tipo Regra Delta com passo $\alpha = 0,09$. É apresentado o par x {0.2, 0.5, 0.4} e y = {0, 1, 0}.

Quais os novos valores das sinapses após o primeiro passo de treinamento?

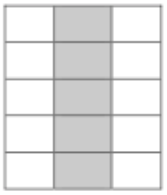


S) Perceptron para representar Dígitos

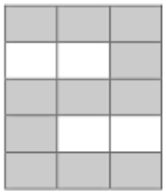
Exemplo: Modelar um perceptron capaz de representar um dispositivo que distinga os dígitos numéricos pares dos ímpares

Resolução:

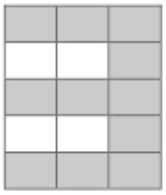
a. Codificação das entradas como um perceptron deve ser feita através de uma matriz de tamanho 5x3, como uma representação visual dos dígitos:



Dígito 1: Matriz M1



Dígito 2: Matriz M2



Dígito 3: Matriz M_3

É necessário representar todos os dígitos, de 0 a 9, portanto, são 10 matrizes de tamanho 5 X 3, denominadas $M_{i,}$ onde i se refere ao dígito representado e i ∈ [0, 9].

Cada elemento M_{ij}, onde i é a linha e j a coluna, da matriz será representado por um nó de entrada X_i do perceptron. Como a camada de entrada do perceptron é uma fila de nós e não uma tabela, deve-se linearizar as matrizes de forma que cada elemento seja mapeado em um nó de entrada.

Como as entradas e saídas dos perceptron são binárias, por definição, cada célula da matriz M_i é assim codificada: 1, se for preenchida para representar o dígito i, e 0, caso contrário.

Exemplo: matriz M_1 que representa o dígito 1, com 15 nós X_i, i ∈ [0, 15]:

$M_{1,1} = X_1 = 0$	$M_{1,2} = X_6 = 1$	$M_{1,3} = X_{11} = 0$
$M_{2,1} = X_2 = 0$	$M_{2,2} = X_7 = 1$	$M_{2,3} = X_{12} = 0$
$M_{3,1} = X_3 = 0$	$M_{3,2} = X_8 = 1$	$M_{3,3} = X_{13} = 0$
$M_{4,1} = X_4 = 0$	$M_{4,2} = X_9 = 1$	$M_{4,3} = X_{14} = 0$
$M_{5,1} = X_5 = 0$	$M_{5,2} = X_{10} = 1$	$M_{5,3} = X_{15} = 0$

b. Codificação das saídas indicará apenas a paridade do dígito de entrada, ou seja, o nó de saída j assumirá o valor 0 para dígito ímpar e 1 para os pares.

c. Cada elemento X_i vai ser multiplicado pelo peso da sinapse entre o nó i, de entrada, e o nó j, de saída, ou seja, o peso W_{ji}

Fazendo o somatório dos produtos se tem net_j, e:

Se: f(x) >= 0, Y=1, dígito impar

< 0, Y=0, dígito par

Desta forma, um perceptron pode representar uma máquina que reconhece os dígitos pares e ímpares, podendo ser treinada como tal.

d. O treinamento se dá apresentando o dígito já codificado, calcula-se a saída. Havendo necessidade, altera o peso. Exemplo: seja o dígito 3 com os valores de X_i correspondentes e devidamente multiplicados pelos respectivos pesos. Se a saída da rede Y_{rede} >= 0, Y = 1 , Ok!

Se Y_{rede} < 0, Y = 0, significa que é necessário aumentar o peso das sinapses, pois o somatório Y_{rede} está baixo e não atinge o valor mínimo do limiar para que a saída seja 1. A solução é aumentar os valores dos pesos para obter Y_{rede} >= 0.

Então, a regra para o dígito par será: se Y_{rede}>= 0, Y=1, então é necessário diminuir o valor dos pesos.

e. Algoritmo de treinamento será assim definido:

Contador_epoca = 0

enquanto (conta_epoca <= condição_final) faça

padrao = 1

enquanto (houver padrões) faca

1. Apresentar a entrada e calcular a saída
2. se saída OK, ir para 3
3. se saída incorreta

a. se 0, $W_{novo} = W_{anterior} + X_{ij}$

b. se 1, $W_{novo} = W_{anterior} - X_{ij}$

4. padrão:= padrão +1

fimenquanto

conta_epoca:=conta_epoca +1

fimenquanto

Onde:

a) época: é uma apresentação à rede do conjunto de padrões

b) padrão: um par contendo o vetor de Entrada e Saída (E/S). Por exemplo, para o dígito 0 (zero) é constituído pela matriz M_0 linearizada, sendo o vetor de entrada: $X_1 = X_2 = X_3 = X_4 = X_5 = X_6 = X_{10} = X_{11} = X_{12} = X_{13} = X_{14} = X_{15} = 1$ e $X_7 = X_8 = X_9 = 0$. E o vetor de saída contendo apenas Y = 1

Exemplo de implementação: treinar um perceptron para reconhecer a função binária dada pela tabela:

X_0 (Viés)	X_1	X_2	Y		
1	0	0	1	→	Padrão 1
1	0	1	0	→	Padrão 2
1	1	0	1	→	Padrão 3
1	1	1	1	→	Padrão 4

Onde:

Padrão é um par de vetores (entrada, saída).

Neste caso:

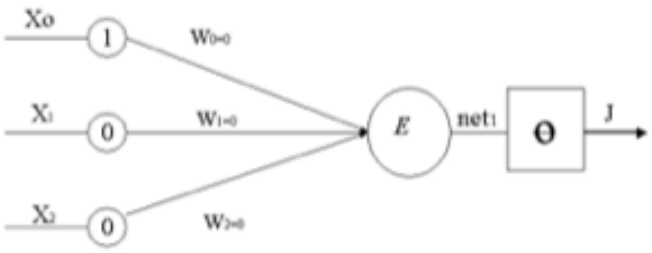
Padrão 1 = vetor de entrada: $[(X_0 = 1), (X_1 = 0), (X_2 = 0)]$ e vetor de saída (Y = 1) (neste caso, de uma só componente).

Solução: sejam 0 (zero) todos os pesos iniciais e $\theta = 1$

Então:

$$\begin{cases} Y = 1 \quad se \quad net_j >= 1 \\ Y = 0 \quad se \quad net_j < 1 \end{cases}$$

<u>1º Padrão:</u>



$=net_1 = W_0X_0 + W_1X_1 + W_2X_2$

$net_1 = 0*1 + 0*0 + 0*0 = 0$

$net_1 = 0 < \theta \rightarrow$ Y = 0, errado, deveria ser1, logo deve-se adicionar a cada peso à sua entrada respectiva.

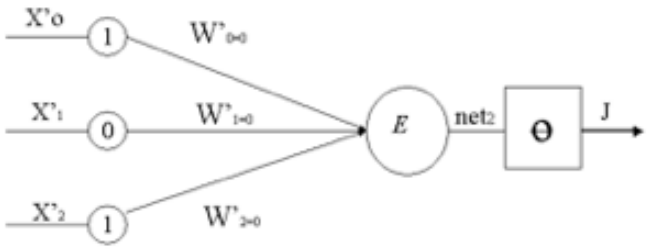
Novos pesos:

$W'_0 = W_0 + X_0 = 0+1 = 1$

$W'_1 = W_1 + X_1 = 0+0 = 0$

$W'_2 = W_2 + X_2 = 0+0 = 0$

2º Padrão:



$net_2 = W'_0X_0 + W'_1X_1 + W'_2X_2$

$net_2 = 1*1 + 0*0 + 0*1 = 1$

$net_2 = 1 = \theta \rightarrow Y = 1$, errado, deveria ser 0, logo deve-se subtrair de cada peso o valor da entrada correspondente.

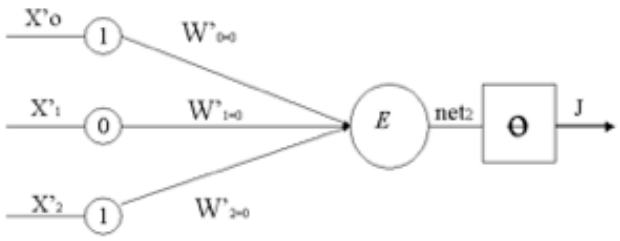
Novos pesos:

$W''_0 = W'_0 - X'_0 = 1-1 = 1$

$W''_1 = W'_1 - X'_1 = 0-0 = 0$

$W''_2 = W'_2 - X'_2 = 0-0 = -1$

3º Padrão:



$net_3 = W_0''X_0'' + W_1''X_1'' + W_2''X_2''$

$net_3 = 0*1 + 0*1 + -1*0 = 0$

$net_3 = 0 < \theta \rightarrow Y = 0$, errado, deveria ser 1, logo deve-se somar a cada peso o valor da entrada correspondente.

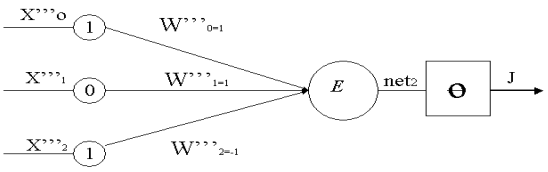
Novos pesos:

$W'''_0 = W''_0 - X''_0 = 0+1 = 1$

$W'''_1 = W''_1 - X''_1 = 0+1 = 1$

$W'''_2 = W''_2 - X''_2 = -1+0 = -1$

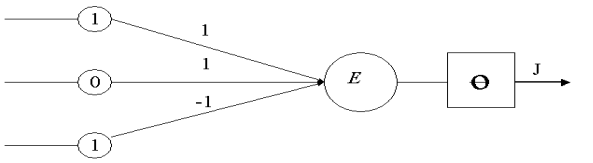
4º Padrão



net4= 1.1+ 1.1+ (-1).1= 1
net4= 1 → J=1, correto.

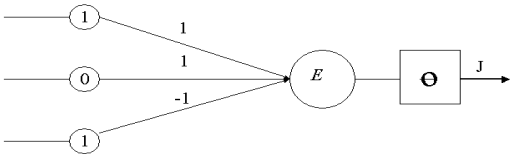
Com isto se finaliza a 1ª época de treinamento e recomeça o processo, que constitui uma nova época.

Teste do 1º Padrão



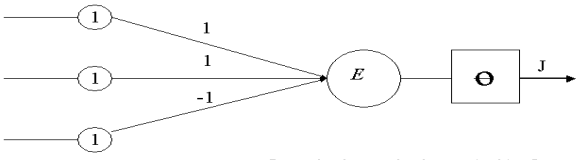
net1= 1.1 + 1.0 + (-1).0 = 1
Logo J = 1, correto

Teste do 2º Padrão



net2= 1.1 + 1.0 + (-1).1 = 0
Logo J = 0, correto

Teste do 3º Padrão



net2= 1.1 + 1.1 + (-1).0 = 0
Logo J = 1, correto

T) Mapas Auto organizáveis – SOM

Exercício Resolvido 1:

Considere uma rede do tipo Kohonen com dois neurônios e os seguintes parâmetros:

Padrão de entrada: X1 = (0,1,0)

$$W=\begin{pmatrix} 0.1 & 0.1 \\ 0.5 & 0.7 \\ 0.3 & 0.6 \end{pmatrix}$$

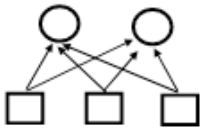
α = 0.8

α (novo)= 0.5 α (antigo)

Vizinhança v = 0

Resolução:

a. Construa a topologia da rede.



b. Treine durante uma época e mostre a atualização dos pesos.

Primeiro Padrão

$Y(1) = (W_{11}\ X_1)2 + (W_{21}\ X_2)2\ (W_{31}\ X_3)2$

$Y(1) = (0.1\ 0)2 + (0.5\ 1)2 + (0.3\ 0)2 = 0.35$

$Y(2) = (W_{12}\ X_1)2 + (W_{22}\ X_2)2\ (W_{32}\ X_3)2$

$Y(2) = (0.1\ 0)2 + (0.7\ 1)2 + (0.6\ 0)2 = 0.4$

Neurônio vencedor Y1 possui a menor distância do padrão a ser representado

$w_{11}(novo) = w_{11}(antigo) + \alpha\ (t)\ (x_1(t) - w_{11}\ (t)$

$w_{11}\ (novo)= 0.1 + 0.8*(0 - 0.1) = 0.02$

$w_{21}(novo) = w_{21}(antigo) + \alpha\ (t)\ (x_2(t) - w_{21}\ (t)\ )$

$w_{21}\ (novo)= 0.5 + 0.8*(1- 0.5) = 0.1$

$w_{31}(novo) = w_{31}(antigo) + \alpha\ (t)\ (x_3(t) - w_{31}\ (t))$

$w_{31}\ (novo)= 0.3 + 0.8*(0 - 0.3) = 0.06$

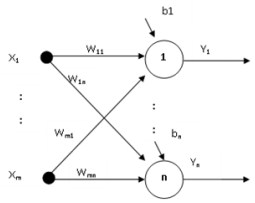
Exercício Resolvido 2:

Realize a simulação das fases de ordenação e convergência de uma rede SOM mostrada na figura 1. Utilize os dados e parâmetros relacionados. A entrada dos dados será feita na sequência em que aparecem no vetor. Desenhe os gráficos do estado da rede no final de cada fase.

Vetores de Entrada:

x1= [-1.0 1.0 -0.5 0.5 0.5]

x2= [0.0 1.0 -0.5 1.0 0.5]



Estado inicial dos neurônios:

wA= [0.0 1.0]

wB= [0.0 -0.5]

wC= [0.5 0.0]

wD= [-0.5 0.0]

Função de Vizinhança: tipo ‘bolha’ com base circular

α = 0.5 (decair 0.1 a cada 2 iterações)

Fase de Ordenação

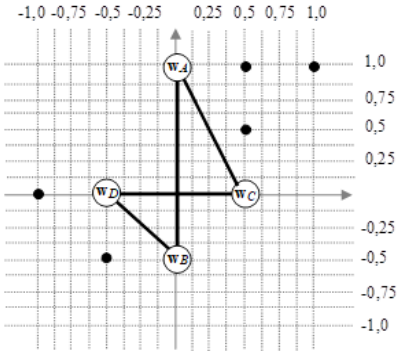
Iterações = 4

Nível de vizinhança no raio <= 1.0

b) Fase de Convergência

Iterações = 4

Nível de vizinhança no raio < 1.0



Estado inicial da rede e espaço de entrada

Fase de Ordenação

Iteração 1) entrada x = [-1.0 0.0]

a. cálculo das distâncias ao quadrado

a1) pesos do neurônio A, wA = [0.0 1.0]

$d_A^2 = (w_{A1} - x_1)^2 + (w_{A2} - x_2)^2$

$d_A^2 = (0.0 - (-1.0))^2 + (1.0 - 0.0)^2 = 1+1 = 2$

a2) pesos do neurônio B, wB = [0.0 -0.5]

$d_B^2 = (w_{B1} - x_1)^2 + (w_{B2} - x_2)^2$

$d_B^2 = (0.0 - (-1.0))^2 + (-0.5 - 0.0)^2 = 1+0.25 = 1.25$

a3) pesos do neurônio C, wC = [0.5 0.0]

$d_C^2 = (w_{C1} - x_1)^2 + (w_{C2} - x_2)^2$

$d_C^2 = (0.5 - (-1.0))^2 + (0.0 - 0.0)^2 = 2.25 = 2.25$

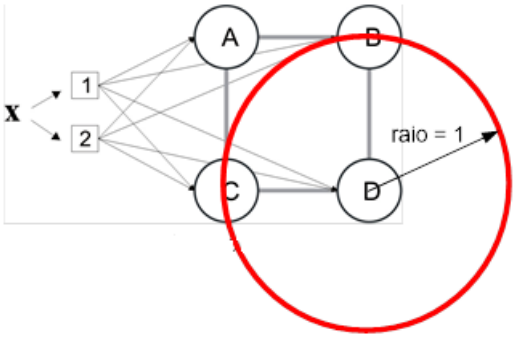
a4) pesos do neurônio D, wD = [-0.5 0.0]

$d_D^2 = (w_{D1} - x_1)^2 + (w_{D2} - x_2)^2$

$d_D^2 = (-0.5 - (-1.0))^2 + (0.0 - 0.0)^2 = 0.5^2 = 0.25$

Menor distância ao quadrado = 0.25, do neurônio D

b) Ajuste de wD, wB e wC, usando vizinhança tipo bolha, com base circular, com raio <= 1.0



$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \alpha(t)h_{vj}(t)[\mathbf{x} - \mathbf{w}_j(t)]$$

$\alpha = 0.5$, h(t) = 1(bolha)

wD = [-0.5 0.0] + 0.5 x 1 x ([-1.0 0.0] - [-0.5 0.0])

= [-0.5 0.0] + 0.5 x 1 x [-0.5 0.0]

= [-0.5 0.0] + [-0.25 0.0]

= [-0.75 0.0]

wC = [0.5 0.0] + 0.5 x 1 x ([-1.0 0.0] - [0.5 0.0])

= [0.5 0.0] + 0.5 x 1 x [-1.5 0.0]

= [0.5 0.0] + [-0.75 0.0]

= [-0.25 0.0]

wB = [0.0 -0.5] + 0.5 x 1 x ([-1.0 0.0] - [0.0 -0.5])

= [0.0 -0.5] + 0.5 x 1 x [-1.0 0.5]

= [0.0 -0.5] + [-0.5 0.25]

= [-0.5 -0.25]

A seguir os resultados da 1ª iteração:

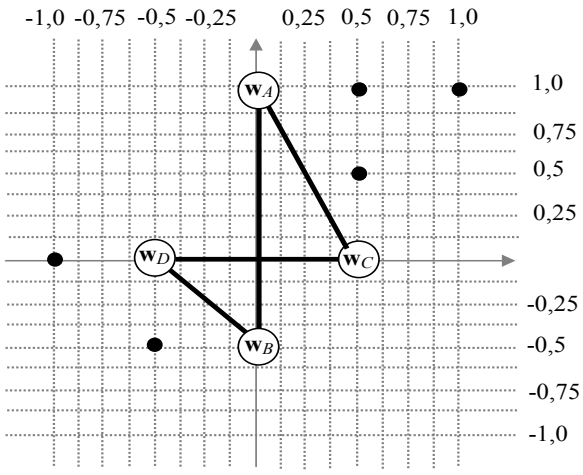
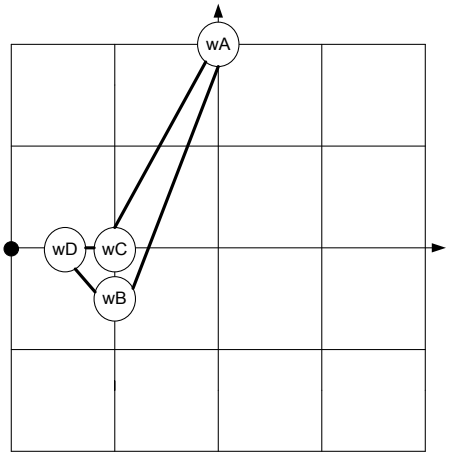


Figura 2 – Estado inicial da rede e espaço de entrada



Fase de ordenação

iteração 2) entrada x = [1.0 1.0]

a. cálculo das distâncias ao quadrado

a1) pesos do neurônio A, wA = [0.0 1.0]

$d_A^2 = (w_{A1} - x_1)^2 + (w_{A2} - x_2)^2$

$d_A^2 = (0.0 - (1.0))^2 + (1.0 - 1.0)^2 = 1+0 = 1$

a2) pesos do neurônio B, wB = [-0.5 -0.25]

$d_B^2 = (w_{B1} - x_1)^2 + (w_{B2} - x_2)^2$

$d_B^2 = (-0.5 - 1.0)^2 + (-0.25 - 1.0)^2 = 2{,}25+1.5625 = 3.8125$

a3) pesos do neurônio C, wC = [-0.25 0.0]

$d_C^2 = (w_{C1} - x_1)^2 + (w_{C2} - x_2)^2$

$d_C^2 = (-0.25 - 1.0)^2 + (0.0 - 1.0)^2 = 1.5625 + 1 = 2.5625$

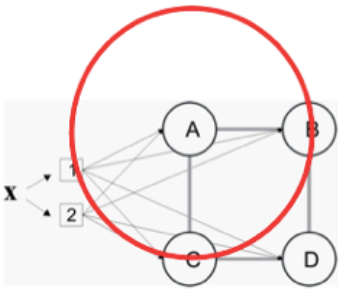
a4) pesos do neurônio D, wD = [-0.75 0.0]

$d_D^2 = (w_{D1} - x_1)^2 + (w_{D2} - x_2)^2$

$d_D^2 = (-0.75 - 1.0)^2 + (0.0 - 1.0)^2 = 3.0625+1 = 4.0625$

Menor distância ao quadrado = 1 do neurônio A

b. Ajuste de wA, wB e wC, usando vizinhança tipo bolha, com base circular, com raio <= 1.0

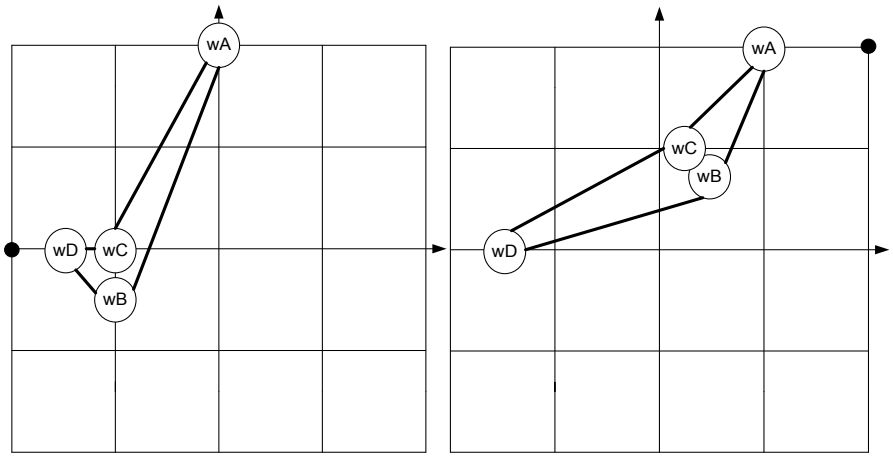


$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \alpha(t) h_{vj}(t)[\mathbf{x} - \mathbf{w}_j(t)]$$

α = 0.5 , h(t) = 1 (bolha)

wA = [0.0 1.0] + 0.5 x 1 x ([1.0 1.0] - [0.0 1.0])
= [0.0 1.0] + 0.5 x 1 x [1.0 0.0]
= [0.0 1.0] + [0.5 0.0]
= [0.5 1.0]
wB = [-0.5 -0.25] + 0.5 x 1 x ([1.0 1.0] - [-0.5 -0.25])
= [-0.5 -0.25] + 0.5 x 1 x [1.5 1.25]
= [-0.5 -0.25] + [0.75 0.625]
= [0.25 0.375]
wC = [-0.25 0.0] + 0.5 x 1 x ([1.0 1.0] - [0.25 0.0])
= [-0.25 0.0] + 0.5 x 1 x [0.75 1.0]
= [-0.25 0.0] + [0.375 0.5]
= [0.125 0.5]

Resultado da 1ª e da 2ª iteração



Fase de ordenação

Iteração 3) entrada x = [-0.5 -0.5]

a. cálculo das distâncias ao quadrado

a1) pesos do neurônio A, wA = [0.5 1.0]

$d_A^2 = (w_{A1} - x_1)^2 + (w_{A2} - x_2)^2$

$d_A^2 = (0.5 - (-0.5))^2 + (1.0 - (-0.5))^2 = 1 + 2.25 = 3.25$

a2) pesos do neurônio B, wB = [0.25 0.375]

$d_B^2 = (w_{B1} - x_1)^2 + (w_{B2} - x_2)^2$

$d_B^2 = (0.25 - (-0.5))^2 + (0.375 - (-0.5))^2 = 0.5625 + 0.7656 = 1.3281$

a3) pesos do neurônio C, wC = [0.125 0.5]

$d_C^2 = (w_{C1} - x_1)^2 + (w_{C2} - x_2)^2$

$d_C^2 = (0.125 - (-0.5))^2 + (0.5 - (-0.5))^2 = 0.3906 + 1 = 1.3906$

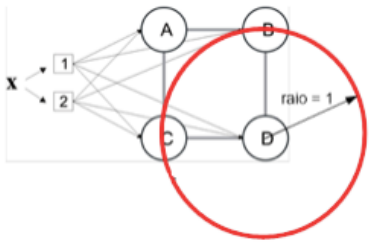
a4) pesos do neurônio D, wD = [-0.75 0.0]

$d_D^2 = (w_{D1} - x_1)^2 + (w_{D2} - x_2)^2$

$d_D^2 = (-0.75 - (-0.5))^2 + (0.0 - (-0.5))^2 = 0.0625 + 0.25 = 0.3125$

Menor distância ao quadrado = 0.3125 do neurônio D

Ajuste de wD, wB e wC, usando vizinhança tipo bolha, com base circular, com raio <= 1.0



$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \alpha(t) h_{vj}(t)[\mathbf{x} - \mathbf{w}_j(t)]$$

α = 0.4 (decai 0.1), h(t) = 1(bolha)

wD = [-0.75 0.0] + 0.4 x 1 x ([-0.5 -0.5] - [-0.75 0.0])

= [-0.75 0.0] + 0.4 x 1 x [0.25 -0.5]

= [-0.75 0.0] + [0.1 -0.2]

= [-0.65 -0.2]

wB = [0.25 0.375] + 0.4 x 1 x ([-0.5 -0.5] - [0.25 0.375])

= [0.25 0.375] + 0.4 x 1 x [-0.75 -0.875]

= [0.25 0.375] + [-0.3 -0.35]

= [-0.05 0.025]

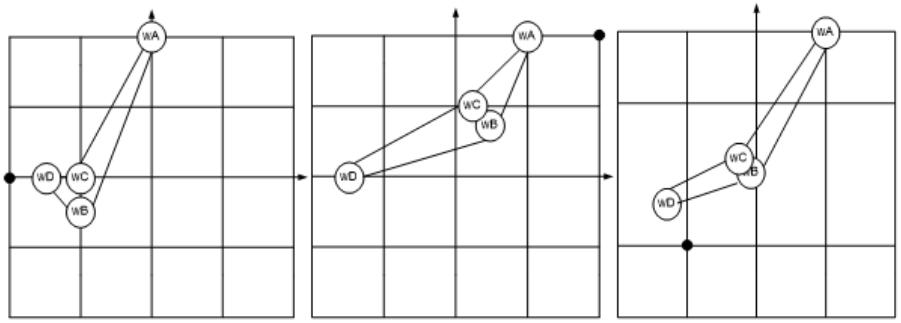
wC = [0.125 0.5] + 0.4 x 1 x ([-0.5 -0.5] - [0.125 0.5])

= [0.125 0.5] + 0.4 x 1 x [-0.625 -1.0]

= [0.125 0.5] + [-0.25 -0.4]

= [-0.125 0.1]

Resultado da 1ª, 2ª e 3ª iteração



Fase de ordenação

Iteração 4) entrada x = [0.5 1.0]

a1) pesos do neurônio A, wA = [0.5 1.0]

a2) pesos do neurônio B, wB = [-0.05 0.025]

a3) pesos do neurônio C, wC = [-0.125 0.1]

a4) pesos do neurônio D, wD = [-0.65 -0.2]

Menor distância ao quadrado = 0 do neurônio A (não foi calculado porque x =

wA).

Ajuste de wA, wB e wC:

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \alpha(t)h_{vj}(t)[\mathbf{x} - \mathbf{w}_j(t)]$$

α = 0.4 , h(t) = 1(bolha)

wA = [0.5 1.0] (não ajusta porque a entrada x coincide com wA)

wB = [-0.05 0.025] + 0.4 x 1 x ([0.5 1.0] - [-0.05 0.025])

= [-0.05 0.025] + 0.4 x 1 x [0.55 0.975]

= [-0.05 0.025] + [0.22 0.39]

= [0.17 0.415]

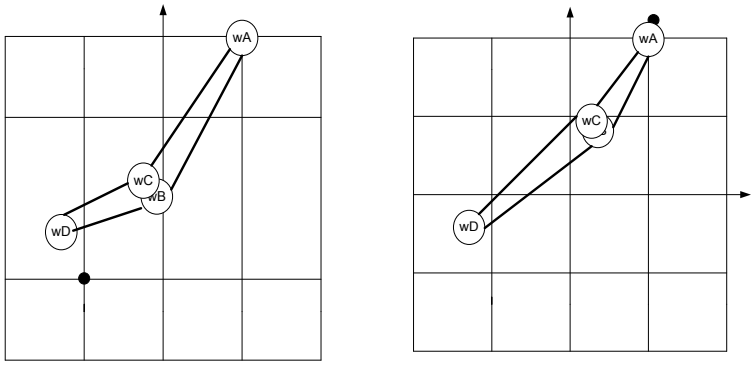
wC = [-0.125 0.1] + 0.4 x 1 x ([0.5 1.0] - [-0.125 0.1])

= [-0.125 0.1] + 0.4 x 1 x [0.625 0.9]

= [-0.125 0.1] + [0.25 0.36]

= [0.125 0.46]

Resultado da 3ª e da 4ª iteração



Fase de convergência

Iteração 5) entrada x = [0.5 0.5]

a1) pesos do neurônio A, wA = [0.5 1.0]

$d_A^2 = (w_{A1} - x_1)^2 + (w_{A2} - x_2)^2$

$d_A^2 = (0.5 - 0.5)^2 + (1.0 - 0.5)^2 = 0 + 0.5 = 0.5$

a2) pesos do neurônio B, wB = [0.17 0.415]

$d_B^2 = (w_{B1} - x_1)^2 + (w_{B2} - x_2)^2$

$d_B^2 = (0.17 - 0.5)^2 + (0.415 - 0.5)^2 = 0.1089 + 0.007225 = 0.1161$

a3) pesos do neurônio C, wC = [0.125 0.46]

$d_C^2 = (w_{C1} - x_1)^2 + (w_{C2} - x_2)^2$

$d_C^2 = (0.125 - 0.5)^2 + (0.46 - 0.5)^2 = 0.1406 + 0.04 = 0.1806$

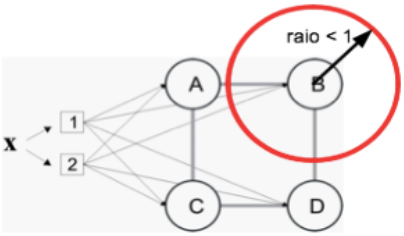
a4) pesos do neurônio D, wD = [-0.65 -0.2]

$d_D^2 = (w_{D1} - x_1)^2 + (w_{D2} - x_2)^2$

$d_D^2 = (-0.65 - 0.5)^2 + (-0.2 - 0.5)^2 = 1.3225 + 0.49 = 1.8125$

Menor distância ao quadrado = 0.1161 do neurônio B

Ajuste wB usando vizinhança tipo bolha, com base circular, com raio < 1.0



$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \alpha(t)h_{vj}(t)[\mathbf{x} - \mathbf{w}_j(t)]$$

α = 0.3 (decai 0.1), h(t) = 1(bolha)

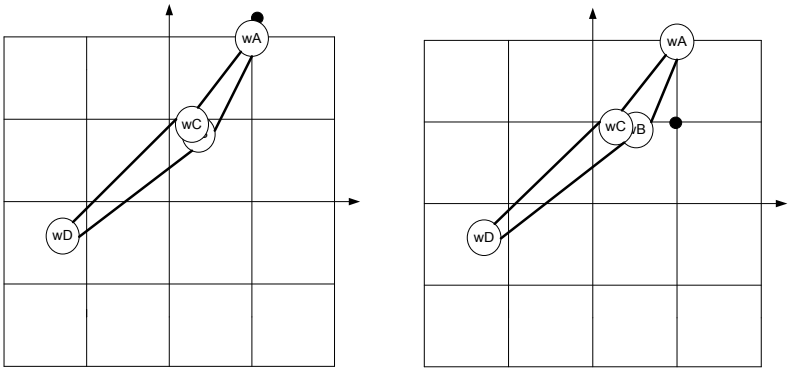
wB = [0.17 0.415] + 0.3 x 1 x ([0.5 0.5] - [0.17 0.415])

= [0.17 0.415] + 0.3 x 1 x [0.33 0.085]

= [0.17 0.415] + [0.099 0.0255]

= [0.269 0.4405]

Resultado da 4ª e da 5ª iteração



Fase de convergência

Iteração6) entrada x = [-1.0 0.0]

a1) pesos do neurônio A, wA = [0.5 1.0]

$$d_A^2 = (w_{A1} - x_1)^2 + (w_{A2} - x_2)^2$$

$$d_A^2 = (0.5 - (-1.0))^2 + (1.0 - 0.0)^2 = 2.25 + 1 = 3.25$$

a2) pesos do neurônio B, wB = [0.269 0.4405]

$$d_B^2 = (w_{B1} - x_1)^2 + (w_{B2} - x_2)^2$$

$$d_B^2 = (0.17 - (-1.0))^2 + (0.415 - 0.0)^2 = 1.3689 + 0.1722 = 1.5411$$

a3) pesos do neurônio C, wC = [0.125 0.46]

$$d_C^2 = (w_{C1} - x_1)^2 + (w_{C2} - x_2)^2$$

$$d_C^2 = (0.125 - (-1.0))^2 + (0.46 - 0.0)^2 = 1.2656 + 0.2116 = 1.4772$$

a4) pesos do neurônio D, wD = [-0.65 -0.2]

$$d_D^2 = (w_{D1} - x_1)^2 + (w_{D2} - x_2)^2$$

$$d_D^2 = (-0.65 - (-1.0))^2 + (-0.2 - 0.0)^2 = 0.1225 + 0.04 = 0.1625$$

Menor distância ao quadrado = 0.1625 do neurônio D

Ajuste wD usando vizinhança tipo bolha, com base circular, com raio < 1.0

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \alpha(t) h_{vj}(t)[\mathbf{x} - \mathbf{w}_j(t)]$$

α = 0.3 , h(t) = 1(bolha)

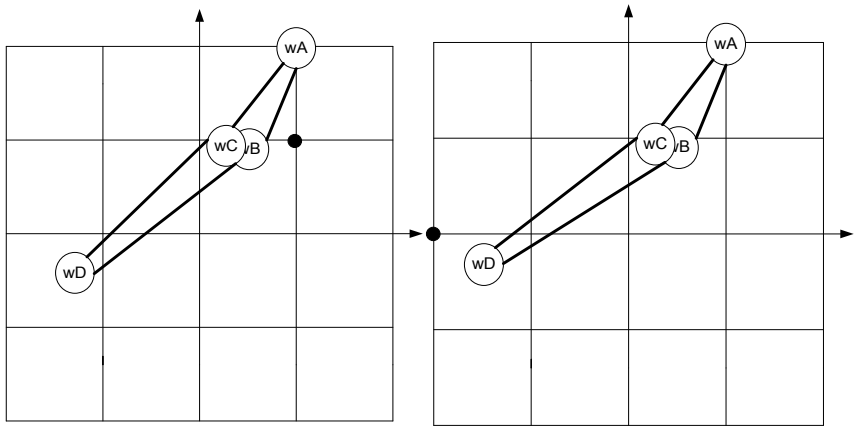
wD = [-0.65 -0.2] + 0.3 x 1 x ([-1.0 0.0] - [-0.65 -0.2])

= [-0.65 -0.2] + 0.3 x 1 x [-0.35 0.2]

= [-0.65 -0.2] + [-0.105 0.06]

= [-0.755 -0.14]

Resultado da 5ª e da 6ª iteração



Fase de convergência

Iteração 7) entrada x = [1.0 1.0]

a1) pesos do neurônio A, wA = [0.5 1.0]

$d_A^2 = (w_{A1} - x_1)^2 + (w_{A2} - x_2)^2$

$d_A^2 = (0.5 - 1.0)^2 + (1.0 - 1.0)^2 = 0.25 + 0 = 0.25$

a2) pesos do neurônio B, wB = [0.269 0.4405]

$d_B^2 = (w_{B1} - x_1)^2 + (w_{B2} - x_2)^2$

$d_B^2 = (0.17 - 1.0)^2 + (0.415 - 1.0)^2 = 0.6889 + 0.3422 = 1.0311$

a3) pesos do neurônio C, wC = [0.125 0.46]

$d_C^2 = (w_{C1} - x_1)^2 + (w_{C2} - x_2)^2$

$d_C^2 = (0.125 - 1.0)^2 + (0.46 - 1.0)^2 = 0.7656 + 0.2916 = 1.0572$

a4) pesos do neurônio D, wD = [-0.755 -0.14]

$d_D^2 = (w_{D1} - x_1)^2 + (w_{D2} - x_2)^2$

$d_D^2 = (-0.755 - 1.0)^2 + (-0.14 - 1.0)^2 = 3.080 + 1.2996 = 4.3796$

Menor distância ao quadrado = 0.25 do neurônio A

Ajuste wA usando vizinhança tipo bolha, com base circular, com raio < 1.0

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \alpha(t) h_{vj}(t)[\mathbf{x} - \mathbf{w}_j(t)]$$

α = 0.2 (decai 0.1) , h(t) = 1(bolha)

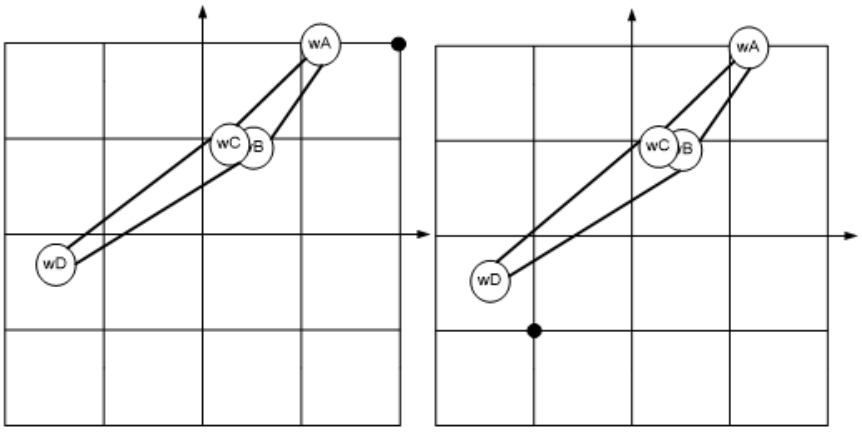
wA = [0.5 1.0] + 0.2 x 1 x ([1.0 1.0] - [0.5 1.0])

= [0.5 1.0] + 0.2 x 1 x [0.5 0.0]

= [0.5 1.0] + [0.1 0.0]

= [0.6 1.0]

Resultado da 6ª e da 7ª iteração



Fase de convergência

Iteração 8) entrada x = [-0.5 -0.5]

a1) pesos do neurônio A, wA = [0.6 1.0]

$d_A^2 = (w_{A1} - x_1)^2 + (w_{A2} - x_2)^2$

$d_A^2 = (0.6 - (-0.5))^2 + (1.0 - (-0.5))^2 = 1.21 + 2.25 = 3.46$

a2) pesos do neurônio B, wB = [0.269 0.4405]

$d_B^2 = (w_{B1} - x_1)^2 + (w_{B2} - x_2)^2$

$d_B^2 = (0.17 - (-0.5))^2 + (0.415 - (-0.5))^2 = 0.4489 + 0.8372 = 1.286$

a3) pesos do neurônio C, wC = [0.125 0.46]

$d_C^2 = (w_{C1} - x_1)^2 + (w_{C2} - x_2)^2$

$d_C^2 = (0.125 - (-0.5))^2 + (0.46 - (-0.5))^2 = 0.3906 + 0.9216 = 1.3122$

a4) pesos do neurônio D, wD = [-0.755 -0.14]

$d_D^2 = (w_{D1} - x_1)^2 + (w_{D2} - x_2)^2$

$d_D^2 = (-0.755 - (-0.5))^2 + (-0.14 - (-0.5))^2 = 0.065 + 0.1296 = 0.1946$

Menor distância ao quadrado = 0.1946 do neurônio D

Ajuste wD usando vizinhança tipo bolha, com base circular, com raio < 1.0

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \alpha(t) h_{ij}(t)[\mathbf{x} - \mathbf{w}_j(t)]$$

$\alpha = 0.2$, h(t) = 1(bolha)

wD = [-0.755 -0.14] + 0.2 x 1 x ([-0.5 -0.5] - [-0.755 -0.14])

= [-0.755 -0.14] + 0.2 x 1 x [0.255 -0.36]

= [-0.755 -0.14] + [0.051 -0.072]

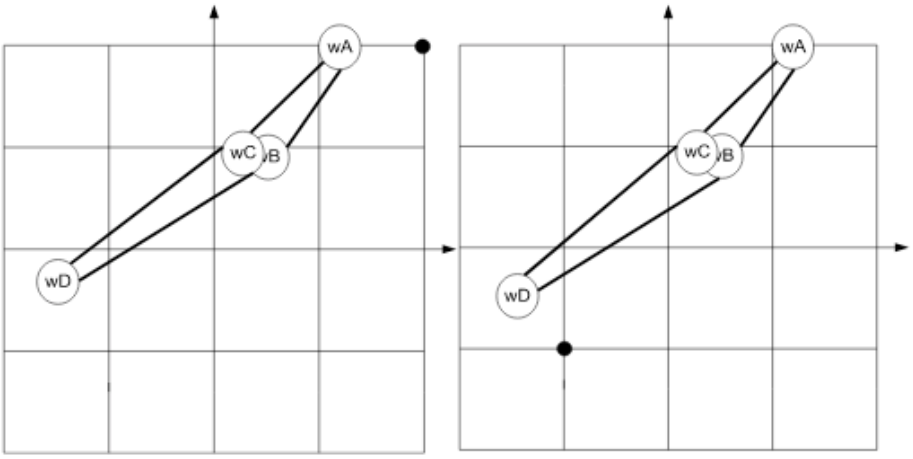
= [-0.704 -0.212]

α = 0.2, h(t) = 1(bolha)

wD = [-0.755 -0.14] + 0.2 x 1 x ([-0.5 -0.5] - [-0.755 -0.14])

= [-0.755 -0.14] + 0.2 1 X ([

Resultado da 7ª e da 8ª iteração



RNA´S - DESENVOLVIMENTO DE APLICAÇÕES

COLETA DE DADOS E SEPARAÇÃO EM CONJUNTOS

Os passos iniciais do processo de desenvolvimento de RNA´s são a coleta de dados relativos ao problema e a sua separação em um conjunto de treinamento e outro de testes. Esta tarefa requer uma análise cuidadosa sobre o problema para minimizar ambigüidades e erros nos dados. Além disso, os dados coletados devem ser significativos e cobrir amplamente o domínio do problema; não devem cobrir apenas as operações normais ou rotineiras, mas também as exceções e as condições limites do domínio do problema.

Normalmente, os dados coletados são separados em duas categorias: dados de treinamento, que serão utilizados para o treinamento da rede e dados de teste, que serão utilizados para verificar sua performance sob condições reais de utilização. Além dessa divisão, pode-se usar também uma subdivisão do conjunto de treinamento, criando um conjunto de validação, utilizado para verificar a eficiência da rede quanto a sua capacidade de generalização durante o treinamento.

Depois de determinados estes conjuntos, eles são, geralmente, colocados em ordem aleatória para prevenção de tendências associadas à ordem de apresentação dos dados. Além disso, pode ser necessário pré-processar estes dados, através de normalizações, escalonamentos e conversões de formato para torná-los mais apropriados à sua utilização na rede.

CONFIGURAÇÃO DA REDE

Neste passo define-se a configuração da RNA, que pode ser dividido nas etapas:

1) Seleção do paradigma neural apropriado à aplicação;

2) Determinação da topologia da rede a ser utilizada - o número de camadas, o número de unidades em cada camada etc;

3) Determinação de parâmetros do algoritmo de treinamento e funções de ativação. Este passo tem um grande impacto na performance do sistema resultante;

Existem metodologias, "dicas" e "truques" na condução destas tarefas - normalmente de forma empírica. A definição da configuração de redes neurais é ainda considerada uma arte, que requer grande experiência dos projetistas.

TREINAMENTO

Neste passo é realizado o treinamento da rede, seguindo o algoritmo escolhido, ajustando os pesos das conexões. É importante considerar, nesta fase, alguns aspectos tais como a inicialização da rede, o modo de treinamento e o tempo de treinamento.

Uma boa escolha dos valores iniciais dos pesos da rede pode diminuir o tempo necessário para o treinamento. Normalmente, os valores iniciais dos pesos da rede são números aleatórios uniformemente distribuídos, em um intervalo definido.

Quanto ao tempo de treinamento, vários fatores podem influenciar a sua duração, porém sempre será necessário utilizar algum critério de parada. O critério de parada do algoritmo backpropagation não é bem definido, sendo geralmente utilizado um número máximo de ciclos. Mas, devem ser consideradas a taxa de erro médio por ciclo e sua capacidade de generalização. Pode ocorrer que em um determinado instante do treinamento a generalização comece a degenerar, causando o problema de overtraining, ou seja, a rede se especializa no conjunto de dados do treinamento e perde a capacidade de generalização.

O treinamento deve ser interrompido quando a rede apresentar boa capacidade de generalização e a taxa de erro for menor que o erro admissível. Assim, deve-se encontrar um ponto ótimo de parada com erro mínimo e capacidade de generalização máxima.

TESTE

Durante esta fase o conjunto de teste é utilizado para determinar a performance da rede com dados que não foram previamente utilizados. Esta performance medida é uma boa indicação de sua performance real.

Devem ser considerados ainda outros testes como análise do comportamento da rede utilizando entradas especiais e análise dos pesos atuais da rede, pois se existirem valores muito pequenos, as conexões associadas podem ser consideradas insignificantes e assim serem eliminadas (prunning). De modo inverso, valores substantivamente maiores que os outros poderiam indicar que houve over-training da rede.

INTEGRAÇÃO

Finalmente, com a rede treinada e avaliada, ela pode ser integrada em um sistema do ambiente operacional da aplicação. Para maior eficiência da solução, este sistema deverá conter facilidades de utilização como interface conveniente e facilidades de aquisição de dados através de planilhas eletrônicas, interfaces com unidades de processamento de sinais, ou arquivos padronizados. Uma boa documentação do sistema e o treinamento de usuários são necessários para o sucesso do mesmo. Além disso, o sistema deve periodicamente monitorar sua performance e fazer a manutenção da rede quando for necessário ou indicar aos projetistas a necessidade de retreinamento. Outras melhorias poderão ainda ser sugeridas quando os usuários forem se tornando mais familiarizados com o sistema, estas sugestões poderão ser muito úteis em novas versões ou em novos produtos.

REFERÊNCIAS

Arbib, M.A., The Handbook of Brain Theory and Neural Networks, The MIT Press, 2nd. Edition, 2002.

Bishop, C. M., Neural Networks for Pattern Recognition, Oxford University Press, 1995.

Braga, A.P., de Carvalho, A.P.L.F., Ludermir, T.B. Redes Neurais Artificiais – Teoria e Aplicações, Editora LTC, 2ª. edição, 2007.

Chauvin, Y., Rumelhart, D.E., Backpropagation: Theory, Architectures, and Applications, Lawrence Erlbaum Associates, 1995.

da Silva, I.N., Spatti, D.H. & Flauzino, R.A., Redes Neurais Artificiais Para Engenharia e Ciências Aplicadas, Artliber Editora Ltda, 2010.

Edelman, G.M., Neural Darwinism: The Theory of Neuronal Group Selection, Basic Books, 1988.

Faceli, K., Lorena, A.C., Gama, J., Carvalho, A.P.L.F., Inteligência Artificial: Uma Abordagem de Aprendizado de Máquina, LTC, Rio de Janeiro, 2011.

Fausett, L., Fundamentals of Neural Networks: Architectures, Algorithms, and Applications, Dorling Kindersley India, 2004.

Gardner, H., Frames of Mind: The Theory of Multiple Intelligences, 3rd edition, BasicBooks, 2011.

Gupta, Madan M.e Rao, Dandina H. - Neuro-Control Systems. Um volume selecionado reeditado. IEEE Neural Networks Council, Sponsor.

Hassoun, M., Fundamentals of Artificial Neural Networks, A Bradford Book, 2003.

Hastie, T., Tibshirani, R., Friedman, J.H., The Elements of Statistical Learning, Springer, 2001.

Haykin, S., Redes Neurais – Princípios e Práticas, 2ª Ed., Editora Bookman, Porto Alegre, 2001.

Hebb, D. O., Wiley, J., Sons, Inc. The Organization of Behavior: A Neuropsychological Theory, New York, 1949.

Hecht-Nielsen, R., Neurocomputing, Addison-Wesley Publishing Co, 1990.

Honavar, V., Uhr, L., Artificial Intelligence and Neural Networks. Academic Press. Boston, EUA, 1995.

Kohonen, T., Learning Vector Quantization. In Handbook of Brain Theory and Neural Networks (M. Arbib, Ed.). MIT Press. Cambridge, EUA. pp 537-540, 1995.

Kohonen, T., Self-Organization and Associative Memory, 3rd edition, Springer-Verlag, 1989.

Kohonen, T., Self-Organizing Maps, 3rd Edition, Springer, 2000.

Kovács, Z. L. - Redes Neurais Artificias. 2ª Edição, Editora Collegium Cognitio, 1996.

Luenberger, D.G., Linear and Nonlinear Programming, 2nd edition, Addison-Wesley, 1984.

Machado, A., Haertel, L. M., Neuroanatomia Funcional. 2a ed. Atheneu. São Paulo, 2002.

Mackay, D.J.C., Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003.

Maren, A., Harston, C., e Pap, R., Handbook of Neural Computing Applications. Academic Press. San Diego, EUA, 1990.

McCulloch, W. S., W. Pitts, W., A Logical Calculus of the Ideas Immanent in Nervous Activity. The bulletin of Mathematical Biophysics, 5(4):115–133, 1943.

Miikkulainen, R., Script-Based Inference and Memory Retrieval in Subsymbolic Story Processing. Applied Intelligence 5. pp 137-163, 1995.

Minsky, M., A Framework for Representing Knowledge. In Mind Design (J. Haugeland, ed.), 1975.

Minsky, M., A Neural-Analogue Calculator Based Upon a Probability Model of Reinforcement. Harvard University Pychological Laboratories Internal Report, 1952.

Minsky, M., Papert, S., Perceptrons. An Introduction to Computational Geometry.. M.I.T. Press, Cambridge, Mass., 1969.

Minsky, M.L., Papert, S.A, Perceptrons: Introduction to Computational Geometry, Expanded edition, The MIT Press, 1988. (1st edition: 1969)

Minsky, M.L., The Society of Mind, Simon & Schuster, 1988.

Mitchell, T.M. Machine Learning, McGraw-Hill, 1997.

Ripley, B. D., Pattern Recognition and Neural Networks. Cambridge University Press. Cambridge, Grã-Bretanha, 1996.

Ritter, H., Self-Organizing Feature Maps: Kohonen Maps. In Handbook of Brain Theory and Neural Networks (M. Arbib, Ed.). MIT Press. Cambridge, EUA. pp 846-851, 1995.

Rosenblatt, F., Principles of Neurodynamics. Spartan. Nova Iorque, EUA, 162.

Rosenblatt, F.. The Perceptron, a Perceiving and Recognizing Automaton Project Para. Cornell Aeronautical Laboratory, 1957.

Rumelhart, D. E., Hinton, G. E., Williams, R. J., Learning Internal Representations by Error Propagation. In Parallell Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1, Foundations. MIT Press. Cambridge, EUA. pp 318-362, 1982.

Rumelhart, D. E., Smolensky, P., McClelland, J. L., e Hinton, G. E., Schemata and Sequential Thought in PDP Models. In Parallell Distributed Processing: Explorations in the Microstructure of Cognition (J. L. McClelland, D. E. Rumelhart and the PDP Research Group, Eds.), vol. 2, Psychological and Biological Models. MIT Press. Cambridge, EUA. pp 7-57, 1986.

Rumelhart, D. E., Zipser, D., Competitive Learning. In Parallell Distributed Processing: Explorations in the Microstructure of Cognition (D. E. Rumelhart, J. L. McClelland and the PDP Research Group, Eds.), vol. 1, Foundations. MIT Press. Cambridge, EUA. pp 151-193, 1986.

Rumelhart, D.E. & McClelland, J.L., Parallel Distributed Processing: Explorations in the Microstructure of Cognition, volumes 1 & 2. The MIT Press, 1986.

Schank, R., Abelson, R., Scripts, Plans, Goals and Understanding. Lawrence Erlbaum Associates. Hillsdale, NJ, EUA, 1977.

Shastri, L., Structured Connectionist Models. In Handbook of Brain Theory and Neural Networks (M. Arbib, Ed.). MIT Press. Cambridge, EUA. pp 949-952, 1995.

Smolensky, P., Information Processing in Dynamical Systems: Foundations of Harmony Theory. In Parallell Distributed Processing: Explorations in the Microstructure of Cognition (D. E. Rumelhart, J. L. McClelland and the PDP Research Group, Eds.), vol. 1, Foundations. MIT Press. Cambridge, EUA. pp 194-281, 1986.

Smolensky, P., Tensor Product Variable Binding and the Representation of Symbolic Structures in

Connectionist Systems. Artificial Intelligence 46. pp 5-46, 1990.

Sobotta, J.; Paulsen, F.; WASCHKE, J. (Org.), Sobotta: Atlas de Anatomia Humana. 23ª. Edição, Guanabara Koogan, Rio de Janeiro, 2012.

Sontag, E. D., Automata and Neural Networks. In Handbook of Brain Theory and Neural Networks (M. Arbib, Ed.). MIT Press. Cambridge, EUA. pp 119-122, 1995.

Sun, R., Schemas, Logics, and Neural Assemblies. Applied Intelligence 5. pp 83-102, 1995.

Touretzky, D., BoltzCONS: Dynamic Symbol Structures in a Connectionist Network. Artificial Intelligence 46. pp 5-46, 1990.

Wasserman, P., Neural Computing - Theory and Practice, Van Nostrand Reinhold - NY, 1989.

SOBRE A AUTORA

MARIA INÊS VASCONCELLOS FURTADO Graduada em Engenharia Química pela Universidade Federal Fluminense (UFF), com mestrado e doutorado em Sistemas Computacionais pela COPPE/UFRJ.

Atualmente é coordenadora do Curso de Engenharia de Produção do Unilasalle-RJ, em Niterói, além de lecionar nos cursos de Engenharia e Sistemas de Informação desta instituição.

Foi Gerente Acadêmica Adjunta, na Universidade Estácio de Sá, Campus Niterói/RJ, além de atuar como docente nos cursos de Análise de Sistemas e Engenharia.

Tem experiência na área de Ciência da Computação, atuando principalmente nas áreas de Lógica e Linguagens de Programação, Redes Neurais Artificiais, Inteligência Artificial e Text Mining. Como docente ministra também aula de Cálculo Diferencial e Integral.

Trabalha como docente de ensino superior desde 1998 e, por anos ministrou aula de Inteligência Artificial a nível de graduação. Em muitos momentos se dedicou a ensinar os fundamentos básicos das Redes Neurais Artificiais, despertando interesse em seus alunos com essa sua paixão.

Agência Brasileira do ISBN
ISBN 978-85-7247-326-2