INTELIGÊNCIA ARTIFICIAL

Prof. Luciano Frontino de Medeiros

TEMA 1 – REDES NEURAIS ARTIFICIAIS

O trabalho em Redes Neurais Artificiais (RNA) tem sido motivado desde o começo pelo reconhecimento de que o cérebro humano processa informações de uma forma inteiramente diferente de um computador convencional. O cérebro é um computador altamente complexo, não linear e paralelo. Possui a capacidade de organizar seus constituintes estruturais, os neurônios, de forma a realizar certos processamentos, tais como reconhecimento de padrões, percepção e controle motor, de forma mais rápida que o mais rápido computador existente (Haykin, 2001, p. 27).

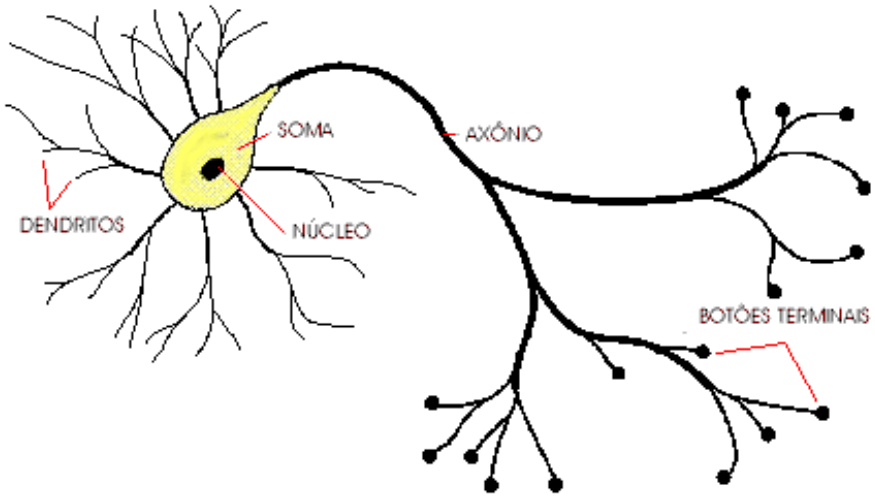
Considere um exemplo da natureza: o sonar de um morcego é um sistema ativo de localização por eco. Além de fornecer informações sobre a distância até o alvo, o sonar transmite também informação sobre a velocidade relativa, tamanho, elevação e outras características do alvo. Toda a complexa computação ocorrendo num cérebro do tamanho de uma ameixa (Haykin, 2001, p. 27).

Os neurônios são as unidades fundamentais dos tecidos do sistema nervoso, incluindo o cérebro. Cada neurônio consiste de um corpo celular, também designado como soma, o qual contém um núcleo. Partindo do corpo da célula existem um número de filamentos denominados dendritos, e um filamento mais longo que é denominado *axônio* (Figura 1). Os dendritos ligam-se ao redor da célula a outras células, e o axônio faz uma conexão mais longa. A essas conexões dá-se o nome de *sinapses*.

O sinal de uma célula a outra se faz mediante uma complicada reação eletroquímica. Substâncias químicas transmissoras são lançadas das sinapses e entram pelos dendritos, aumentando ou baixando o potencial elétrico do corpo da célula. Quando o potencial chega a um limiar, um pulso elétrico ou potencial de ação é mandado pelo axônio. O pulso espalha-se ao longo das conexões existentes pelo axônio, eventualmente chegando a outras sinapses e lançando transmissores ao corpo de outras células.

Sinapses que incrementam o potencial de outras células são denominadas **excitatórias**, enquanto as que diminuem são denominadas **inibitórias**. Os neurônios podem formar novas conexões com outros neurônios, e por meio de tais mecanismos se forma a base para o aprendizado do cérebro.

Figura 1 – Ilustração de um neurônio biológico



Fonte: adaptado de Medeiros, 2006.

Uma rede neural biológica pode, dessa forma, ser abstraída para simular o seu comportamento. Assim, podemos conceituar uma rede neural artificial como um processador maciça e paralelamente distribuído, constituído de unidades de processamento simples, as quais têm a propensão natural para armazenar conhecimento experimental e torná-lo disponível para o uso (Haykin, 2001).

Uma rede neural artificial tem uma série de propriedades (Haykin, 2001, p. 30):

- **Não linearidade**: neurônios podem ser lineares ou não lineares, permitindo, dessa forma, aproximações robustas de funções de mapeamento que tenham característica não linear.
- **Mapeamento entrada-saída**: a rede aprende a partir de exemplos, estabelecendo mapeamento entre os padrões apresentados na entrada com as saídas dadas pelos exemplos.
- **Adaptabilidade**: redes neurais podem ser treinadas e armazenar o conhecimento nos pesos sinápticos, podendo adaptar-se caso o conjunto de amostras utilizado para o treinamento se modifique ao longo do tempo.
- **Resposta a evidências**: uma rede neural pode perfazer uma tarefa de seleção de um padrão, mas também informar sobre o grau de confiança ou crença referente ao padrão escolhido.
- **Informação contextual**: o conhecimento é armazenado na própria estrutura e pela ativação da rede neural.

- **Tolerância a falhas**: redes que sejam implementadas em hardware são tolerantes a falhas, em caso de neurônios ou conexões que possam ser danificados, ou mesmo em software utilizando técnicas de poda de redes, que reduzem a quantidade de neurônios ou sinapses, mantendo a mesma condição de performance.

Uma rede neural artificial tem a sua construção dependente de alguns elementos:

- **Número de camadas**: as RNAs possuem pelo menos uma camada de entrada, de onde recebe os sinais ou características das amostras, e uma camada de saída que apresenta os padrões ou classes mapeados para os conjuntos de treinamento. Também podem possuir uma ou mais camadas ocultas, como no caso do perceptron multicamada e das redes de base radial.
- **Quantidade de neurônios por camada**: a quantidade de neurônios vai depender da natureza do problema sendo abordado. A camada de entrada terá tantos neurônios conforme as características das amostras do conjunto de treinamento. A camada de saída terá os neurônios referentes às classes a que pertencem as amostras do conjunto de treinamento. A camada oculta ou escondida pode ter a quantidade de neurônios variável, conforme a característica do mapeamento que se deseja.
- **Tipo de função de transferência:** a função de transferência define a ativação do neurônio. Podem ser utilizadas funções discretas (tais como a função degrau, utilizada no perceptron a ser explicado adiante) ou funções contínuas (como a função sigmoide para o perceptron multicamada).
- **Método de treinamento**: ao longo dos anos foram desenvolvidos diversos métodos ou algoritmos de treinamento. Dentre os métodos mais utilizados está o algoritmo de retropropagação (*backpropagation*), que utiliza a informação do erro na atualização dos pesos.

As RNAs podem ser aprendidas de diversas formas:

- **Aprendizagem por correção de erros**: a informação do erro é utilizada para modificar os pesos sinápticos.
- **Aprendizagem baseada em memória**: um grande número de exemplos de entrada e saída são armazenados, e uma amostra é comparada com a sua vizinhança para se identificar a classe à qual pertence.

- **Aprendizagem hebbiana**: baseado nos estudos de Hebb, utiliza uma regra associativa, que aumenta a força dos pesos positivamente correlacionados ou diminui a daqueles negativamente correlacionados.
- **Aprendizagem competitiva**: nesse tipo de aprendizagem os neurônios competem entre si tendo-se um vencedor que estará ativo em um certo instante (aprendizagem denominada também de *winner-takes-all*).
- **Aprendizagem de Boltzmann**: aprendizagem com características estocásticas, derivada das ideias da mecânica estatística. Nesse caso, os neurônios constituem uma estrutura recorrente e operam de maneira binária, controlados por uma função de energia. A atualização dos pesos se dá por correlação, operando em duas condições: uma condição presa (os neurônios visíveis estão presos a estados específicos) e outra condição livre (todos os neurônios podem operar livremente).

A aprendizagem ainda pode ser caracterizada como supervisionada (em que há o feedback que retorna à rede para orientar o treinamento) e não supervisionada (a rede aprende de forma auto-organizada). Quanto às tarefas que podem ser executadas por uma RNA estão:

- Associação de padrões.
- Reconhecimento de padrões.
- Aproximação de funções.
- Controle.
- Filtragem.

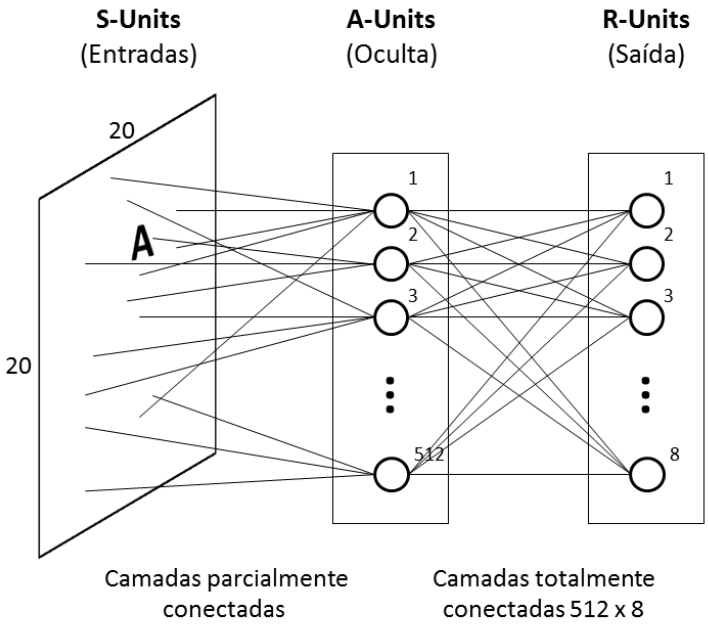
TEMA 2 – O PERCEPTRON

O perceptron é um dispositivo eletrônico inventado em 1957 por Frank Rosenblatt (1928-1971), psicólogo norte-americano, considerado uma espécie de “homem da Renascença, devido à sua excelência em várias áreas, incluindo computação, matemática, neurofisiologia, astronomia e música” (Rosenblatt’s Contributions, [S.d.], tradução nossa). O dispositivo foi construído de acordo com princípios biológicos e mostrava capacidade de aprendizado (Rosenblatt, 1958). Rosenblatt organizou-o em três camadas de unidades (ou neurônios): **sensoriais** (S), **associativas** (A) e **responsivas** (R). A camada de entrada com unidades sensoriais recebe os padrões visuais, a camada com as unidades associativas relaciona as unidades de entrada com as unidades geradoras de saída,

funcionando como uma camada oculta, e a camada com as unidades geradoras de respostas fornecem um resultado de acordo com os padrões apresentados à camada de entrada.

Rosenblatt desenvolveu também um perceptron implementado em hardware com 400 unidades fotossensoras, uma camada de associação de 512 unidades e outra de saída, de oito unidades. Trata-se do protótipo denominado Mark I (Figura 2).

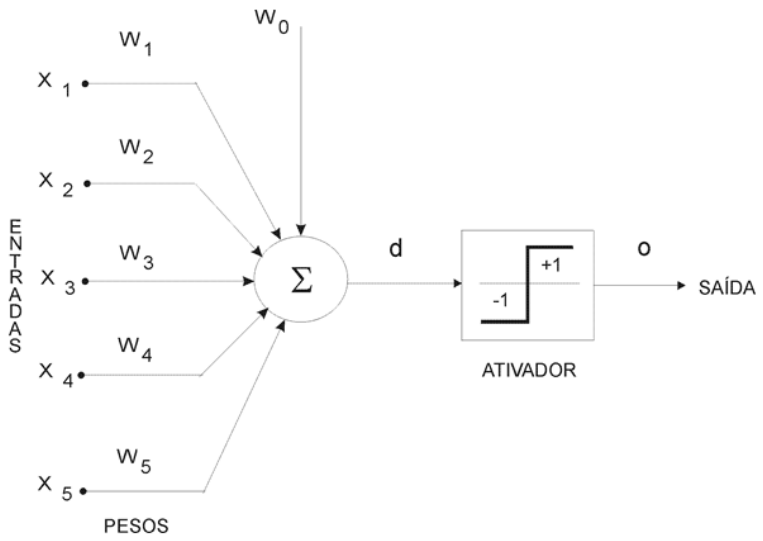
Figura 2 – Ilustração do perceptron de Rosenblatt



Fonte: Medeiros, 2018.

O perceptron obtém os sinais do ambiente por meio das entradas, em que são apresentados os valores correspondentes aos padrões que queremos classificar, como valores de pixels de imagem ou características de um produto. Na figura 3 o perceptron possui cinco entradas.

Figura 3 – Arquitetura de um perceptron. As variáveis significam: x – valores de entrada; w – valores dos pesos; d – saída intermediária; o – saída ativada



Fonte: adaptado de Medeiros, 2006.

Fazendo parte da estrutura interna do perceptron, temos os pesos ou sinapses. Os pesos assumirão valores tais que, quando aplicarmos um padrão na entrada, obteremos uma saída intermediária *d*. O aprendizado da rede ficará armazenado nos pesos e seus valores são obtidos mediante um processo de treinamento. O valor w_0 é chamado de **bias**, sendo fixo e entendido como uma espécie de ajuste fino, o qual não multiplica com entrada nenhuma.

A saída intermediária *d* é calculada mediante o somatório da multiplicação entre cada entrada e seu peso:

$$d = x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4 + x_5w_5 + w_0$$

ou, de forma genérica (com $n = 5$),

$$d = w_0 + \sum_{i=1}^{n} x_i w_i \qquad (1)$$

Após o cálculo da saída intermediária *d*, precisamos então calcular agora a saída ativada o (do inglês, *output*). Como visto na figura 3, a saída ativada *o* depende do resultado da saída intermediária *d*. Caso o resultado obtido em *d* for maior ou igual a 0 (zero), o será igual a +1 (mais um); se *d* for menor que 0 (zero), o será igual a –1 (menos um).

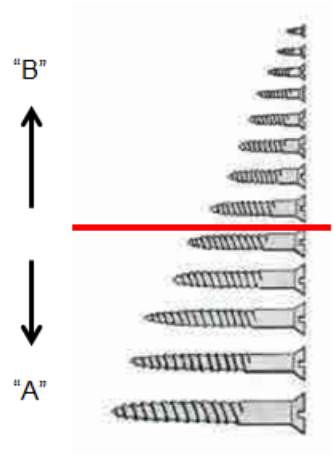
TEMA 3 – DINÂMICA DO PERCEPTRON

Para mostrar o uso do perceptron em uma tarefa simples de classificação, veremos o exemplo da classificação de parafusos. Existem duas classes de parafusos sendo fabricados: a classe “A” e a classe “B”. Os parafusos serão classificados de acordo com o seu comprimento e o diâmetro. Digamos que, em média, os parafusos que tenham tamanho próximo de 5 cm sejam os maiores e os próximos de 1 cm sejam os menores e que temos parafusos de tamanhos diferentes dentro desta faixa. O mesmo pode ser feito com o diâmetro, sendo que parafusos de 5 mm de diâmetro sejam os maiores e 1 mm sejam os menores, havendo tamanhos intermediários. O perceptron terá de dizer se um parafuso com certo comprimento e diâmetro seja pertencente à classe “A” ou “B”.

Utilizando o treinamento por amostragem, na figura 4 temos um conjunto de medidas de parafusos que serão utilizados para “treinar” o perceptron, ou seja, calcular os pesos.

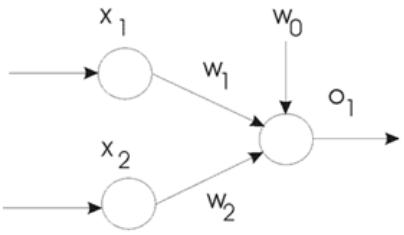
Figura 4 – Tabela com amostras que serão utilizadas para o treinamento do perceptron

Paraf.	x_1	x_2
1	4 cm	3 mm
2	1 cm	1 mm
3	2 cm	2 mm
4	2 cm	3 mm
5	5 cm	3 mm
6	3 cm	2 mm



Na figura 5, temos uma representação simplificada da **arquitetura** (também chamada de **topologia**) desse perceptron, contendo duas unidades de entrada (x_1 e x_2), dois pesos (w_1 e w_2, com um peso fixo w_0) e uma unidade de saída (o_1) para inferir sobre a classe a que pertence o parafuso.

Figura 5 – Topologia do perceptron para a tarefa de classificação de parafusos



O processo de treinamento dessa rede será visualizado mais adiante. Por enquanto, vamos afirmar que esse treinamento já foi feito e os valores de pesos que foram encontrados (exceto w_0, que foi prefixado em -1) estão descritos na tabela 1.

Tabela 1 – Valores dos pesos que satisfazem a classificação do perceptron para o problema dos parafusos

w_1	w_2	w_0
0,21	0,22	-1

Quando alimentamos a rede com os valores das amostras consideradas para o treinamento, vemos que o perceptron classifica corretamente, assumindo que o valor da saída +1 seja atribuído à classe “A” e o valor de saída -1 seja atribuído à classe “B”. O cálculo da variável d_1 utilizou a fórmula genérica (1) vista anteriormente. Pela regra de ativação, se o valor d_1 é positivo, a saída será +1, senão será -1.

Tabela 2 – Classificação após o treinamento

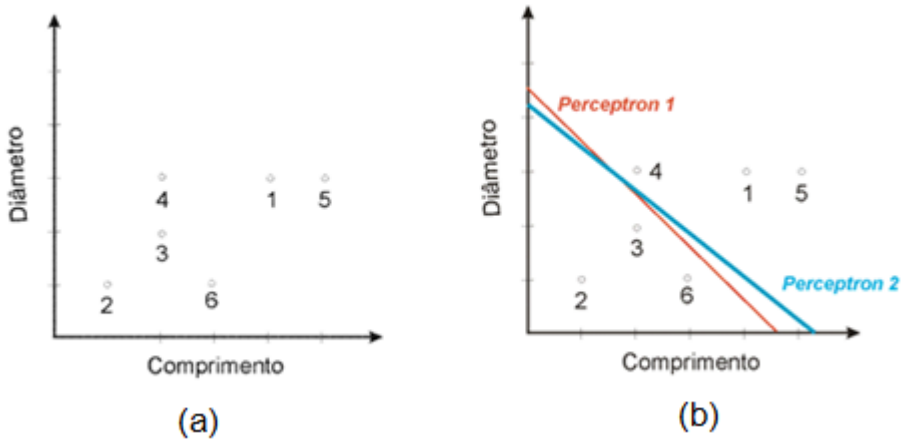
Paraf.	x_1	x_2	d_1	o_1	Classe
1	4 cm	3 mm	0,50	+1	A
2	1 cm	1 mm	-0,57	-1	B
3	2 cm	2 mm	-0,14	-1	B
4	2 cm	3 mm	0,08	+1	A
5	5 cm	3 mm	0,71	+1	A
6	3 cm	2 mm	0,07	+1	A

Se plotarmos os dados em um gráfico de coordenadas cartesianas, com o comprimento x_1 e o diâmetro x_2, pode-se notar que cada parafuso será um ponto representado nesse plano. O processo de classificação com um perceptron simples pode ser visualizado como a separação dos pontos no plano por meio de

uma **reta**. Dessa forma, pode-se constatar na Figura 6 (a) que infinitas retas podem ser traçadas, separando o grupo de pontos A (pontos 1, 4 e 5) do grupo de pontos B (2, 3 e 6).

Isso leva ao conceito de **separabilidade linear**. Para funcionar corretamente, o perceptron deve trabalhar com classes que sejam separáveis linearmente. Os padrões ou amostras pertencentes a uma classe devem estar suficiente separados da outra classe para uma correta classificação. Duas classes são separáveis linearmente se elas podem ser separadas por meio de uma reta, como pode ser ilustrado na figura 6 (b).

Figura 6 – Representação gráfica no plano das amostras conforme as duas entradas do perceptron. Em (b), vemos o que significa a separabilidade linear



Portanto, os valores a serem buscados para o aprendizado do perceptron permitir a obtenção das classes corretas a partir dos valores dos pesos como os da figura 6. O processo de aprendizagem ou treinamento deverá se encarregar de encontrar esses valores para que a rede o classifique corretamente.

TEMA 4 – APRENDIZADO DO PERCEPTRON

Um dos algoritmos mais estudados para o treinamento de um perceptron é o algoritmo do mínimo quadrado médio (em inglês, *Least Mean Square* – LMS), que foi criado por Widrow e Hoff em 1960. Sua implementação não é complexa e permite alcançar bons resultados no treinamento de um perceptron. O algoritmo LMS consiste em calcular o erro de classificação para, na sequência, utilizar uma fração desse erro para o ajuste dos pesos (Medeiros, 2018).

Costuma-se começar o processo de treinamento adotando valores aleatórios para os pesos. Ao proceder a primeira alimentação de valores à

entrada, acontece o erro de classificação após as amostras terem sido apresentadas ao perceptron. Durante o treinamento e de forma gradativa, um pequeno percentual desse erro é propagado de volta ao perceptron. Esse percentual é denominado *taxa de aprendizagem*. No que se refere aos erros, introduziremos dois conceitos: o erro individual e o erro global.

O erro individual refere-se a cada classificação de amostra e tem a seguinte fórmula:

$$e_i = (o_i - f_i) \tag{2}$$

Em que o erro e_i é a diferença entre a saída desejada o_i e a saída calculada após a ativação f_i. O erro global é uma medida de desempenho global do perceptron, sendo a média quadrática dos erros individuais:

$$E = \frac{1}{2}\sum_{i=1}^{n} e_i^2 \tag{3}$$

O aprendizado será gradativo, e atualizará os pesos a cada execução da alimentação à frente. Para calcular a parcela do erro que será realimentado no perceptron, utilizaremos a fórmula do delta:

$$\Delta_j = \eta e_i x_j \tag{4}$$

Em que:

η = taxa de aprendizado;

Δ_j = delta calculado;

e_i = erro ou diferença;

x_j = neurônio de entrada;

n = número de amostras (a seguir).

Os deltas são calculados multiplicando-se o erro de cada amostra pela entrada referente ao delta. Logo após, é feita uma média dos valores calculados para os deltas. Após o cálculo, os pesos são atualizados com os valores dos deltas,

$$w_j(n+1) = w_j(n) + \Delta_j(n) \tag{5}$$

Por essa característica, tal regra de aprendizagem costuma ser chamada de **regra delta**.

O treinamento do perceptron é ilustrado de forma didática nas figuras 7 a 16, com o uso de um simulador escrito em HTML e Javascript. O treinamento começa no passo $n = 1$, com os valores iniciais dos pesos w_1 e w_2 iguais a 1 e o peso bias w_0 fixado em -1.

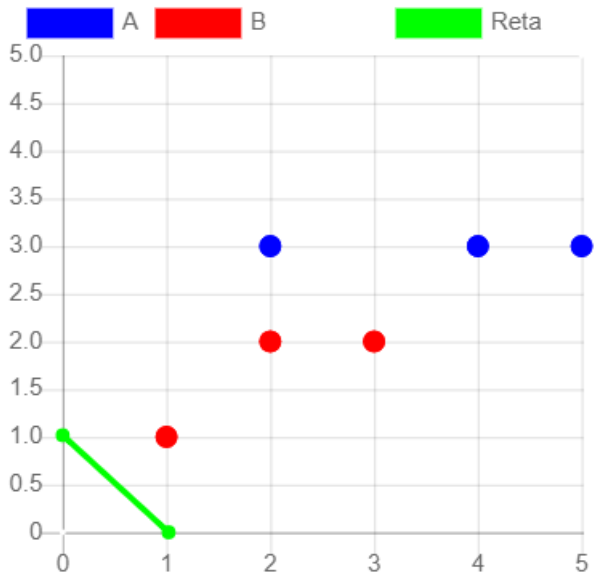
Figura 7 – Estado inicial do treinamento do perceptron. Na parte superior estão os valores da taxa de treinamento ou aprendizado ($\eta = 0.01$ ou 1%); os valores iniciais dos pesos (w_1 e w_2 iguais a 1,0 e w_0 igual a -1); e as amostras com os valores das entradas alimentadas, de acordo com as fórmulas

Simulador de Perceptron Simples

Parâmetros			η	0.01	Passo		0	Limite	100
Pesos			w_1	1.000	w_2	1.000	w_0		-1
No.	x_1	x_2	d	f	o	e	e^2	Δ_1	Δ_2
1	4	3	6.00	1	1	0.00	0.00	0.00	0.00
2	1	1	1.00	1	-1	-2.00	4.00	-0.02	-0.02
3	2	2	3.00	1	-1	-2.00	4.00	-0.04	-0.04
4	2	3	4.00	1	1	0.00	0.00	0.00	0.00
5	5	3	7.00	1	1	0.00	0.00	0.00	0.00
6	3	2	4.00	1	-1	-2.00	4.00	-0.06	-0.04
Tempo(s)			10 ▾	Σ / Δ			12	-0.02	-0.02

Fonte: elaborado base em Frontino ([S.d.]).

Figura 8 – Representação no plano cartesiano do estado inicial do perceptron



As amostras dos parafusos são colocadas linha a linha. A coluna d mostra o cálculo referente à fórmula (1). Por exemplo, na amostra 1 (comprimento de 4 cm e largura de 3 mm) temos $d = x_1 * w_1 + x_2 * w_2 + w_0 = 4 * 1 + 3 * 1 - 1 = 6$. Na coluna f temos a função de ativação: caso d seja maior que zero, assume o valor 1 (positivo); se for menor que zero, assume o valor -1. A coluna o refere-se à classe desejada: caso a amostra faça parte da classe "A", o valor de o deve ser 1; se for da classe "B", deve ser -1. As colunas f e o auxiliam na visualização das classes.

A seguir, tem-se a coluna e que se refere ao erro de classificação, sendo calculada conforme a fórmula (2). Por exemplo, a amostra 2 deveria ser da classe "B", mas está sendo classificada pelo perceptron como classe "A". Então, o erro e é igual à diferença entre a saída desejada o pela calculada f. Assim, $e = o - f = -1 - 1 = -2$ (na tabela, o número negativo aparece entre parênteses e com o fundo em amarelo). Na parte inferior da tabela, há o erro global calculado de acordo com a fórmula (3). Cada erro na planilha é elevado ao quadrado e somado para resultar no erro global E. O erro global E é uma medida de desempenho do sistema. Na medida em que o erro global E tende a zero, tem-se a indicação que o perceptron está aprendendo.

O valor do erro permite agora o cálculo dos deltas parciais, Δ_1 e Δ_2. Para isso, usa-se a fórmula (4). No caso da amostra 2, os valores dos deltas parciais serão: $\Delta_1 = \eta e_1 x_1 = 0.01 * -2{,}0 * 1 = -0.02$; e $\Delta_2 = \eta e_1 x_2 = 0.01 * -2{,}0 * 1 = -0.02$. Pode-se notar que, para aquelas amostras em que a classificação é correta, não há erro e os deltas são iguais a zero. Na parte inferior da tabela, é feito então o cálculo da média de todos os deltas calculados, chegando a um valor de delta para cada entrada: $\Delta_1 = -0.02$ e $\Delta_2 = -0.02$.

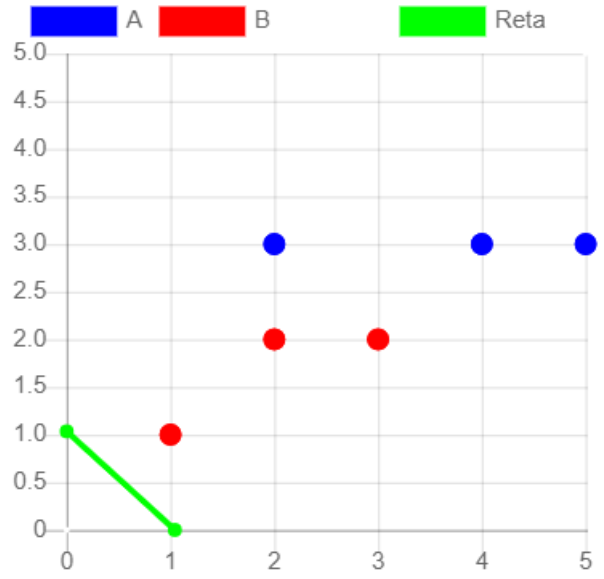
Ao final, com os deltas calculados, utilizamos a fórmula (5) para atualizar o valor dos pesos. Dessa forma, o peso w_1 que será utilizado para o passo n = 2 será reajustado conforme o valor de Δ_1: $w_1(2) = w_1(1) + \Delta_1 = 1 - 0.02 = 0.98$ e $w_2(2) = w_2(1) + \Delta_2 = 1 - 0.02 = 0.98$. Na Figura 9, temos a tabela no passo n = 2, com os valores dos pesos atualizados, com a Figura 10 mostrando a representação no plano cartesiano com os valores dos pesos atualizados e mostrando o posicionamento da reta de classificação.

Figura 9 – Planilha de treinamento do perceptron para classificação de parafusos no passo n = 2. Os valores atualizados dos pesos w_1 e w_2 têm impacto em cascata sobre os valores da planilha para esse passo. Veja que os valores de d_1 já são diferentes daqueles do passo n = 1

Simulador de Perceptron Simples									
Parâmetros			η	0.01	Passo		2	Limite	100
Pesos			w_1	0.960	w_2	0.967	w_0		-1
No.	x_1	x_2	d	f	o	e	e^2	Δ_1	Δ_2
1	4	3	5.87	1	1	0.00	0.00	0.00	0.00
2	1	1	0.96	1	-1	-2.00	4.00	-0.02	-0.02
3	2	2	2.93	1	-1	-2.00	4.00	-0.04	-0.04
4	2	3	3.91	1	1	0.00	0.00	0.00	0.00
5	5	3	6.85	1	1	0.00	0.00	0.00	0.00
6	3	2	3.91	1	-1	-2.00	4.00	-0.06	-0.04
Tempo(s)			10 ▾	Σ / $\overline{\Delta}$			12	-0.02	-0.02

Crédito: elaborado com base em Frontino ([S.d.]).

Figura 10 – Representação no plano cartesiano do estado $n = 2$ do perceptron



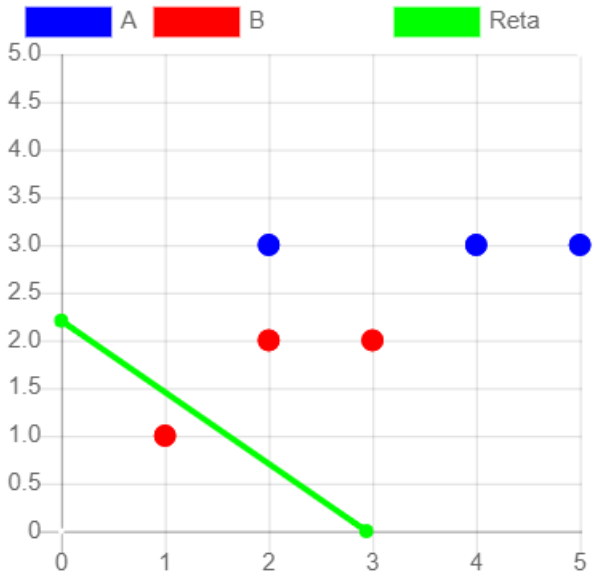
No passo $n = 34$ (figura 11), vemos que os pesos se reduziram a w_1 = 0.36 e w2 = 0.47. Nesse passo, a amostra 2 está sendo classificada corretamente agora para a classe “B”. No entanto, ainda temos as amostras 3 e 6 que estão sendo classificadas incorretamente. Na figura 12, pode-se constatar que a reta “se move” em direção à região limite das duas classes.

Figura 11 – Estado do treinamento do perceptron no passo $n = 34$. Nessa altura do treinamento, a amostra 2 já se encontra classificada corretamente para a classe “B”. O erro global E diminuiu de 12.0 para 8.0

Simulador de Perceptron Simples									
Parâmetros			η	0.01	Passo		34	Limite	100
Pesos			w_1	0.340	w_2	0.453	w_0		-1
No.	x_1	x_2	d	f	o	e	e^2	Δ_1	Δ_2
1	4	3	1.83	1	1	0.00	0.00	0.00	0.00
2	1	1	-0.18	-1	-1	0.00	0.00	0.00	0.00
3	2	2	0.65	1	-1	-2.00	4.00	-0.04	-0.04
4	2	3	1.11	1	1	0.00	0.00	0.00	0.00
5	5	3	2.18	1	1	0.00	0.00	0.00	0.00
6	3	2	1.00	1	-1	-2.00	4.00	-0.06	-0.04
Tempo(s)			10 ▾	$\Sigma / \overline{\Delta}$			8	-0.02	-0.01

Fonte: elaborado com base em Frontino ([S.d.]).

Figura 12 – Representação no plano cartesiano do estado $n = 34$ do perceptron



No passo 48 (figura 13), com os pesos $w_1 = 0.14$ e $w_2 = 0.30$, a amostra 6 ainda está sendo classificada incorretamente. O erro global é $E = 4.0$ mostra que ao longo de todo o treinamento o perceptron foi capaz de aprender a classificar corretamente, de forma gradativa. Pode-se verificar também na figura 14 que a reta está próxima de dividir o espaço das duas classes. Quando se chega ao passo $n = 49$ (figura 15), a amostra 6 tem o valor de d negativo, com os pesos

aproximados $w_1 = 0.13$ e $w_2 = 0.29$; assim, ele classifica agora corretamente na classe “B”. Como não há mais valores de erro, o erro global E é zero e podemos afirmar que o perceptron consegue classificar o conjunto de parafusos nas classes corretas. Isso pode ser visualizado também na figura 16, em que a reta agora divide o espaço entre as classes, de forma que cada amostra esteja classificada corretamente. Isso está relacionado na teoria de redes neurais do que se chama de **teorema de convergência do perceptron**.

Figura 13 – Estado do perceptron no passo $n = 48$. Nesse ponto do treinamento, a amostra 6 ainda é atribuída incorretamente para a classe “A”. O erro global E diminuiu para 4.0

Simulador de Perceptron Simples									
Parâmetros			η	0.01	Passo	48	Limite	100	
Pesos			w_1	0.133	w_2	0.293	w_0	-1	
No.	x_1	x_2	d	f	o	e	e^2	Δ_1	Δ_2
1	4	3	0.47	1	1	0.00	0.00	0.00	0.00
2	1	1	-0.56	-1	-1	0.00	0.00	0.00	0.00
3	2	2	-0.11	-1	-1	0.00	0.00	0.00	0.00
4	2	3	0.19	1	1	0.00	0.00	0.00	0.00
5	5	3	0.62	1	1	0.00	0.00	0.00	0.00
6	3	2	0.03	1	-1	-2.00	4.00	-0.06	-0.04
Tempo(s)			10 ▾	Σ / $\overline{\Delta}$			4	-0.01	-0.01

Fonte: elaborado com base em Frontino ([S.d.]).

Figura 14 – Representação no plano cartesiano do estado $n = 48$ do perceptron

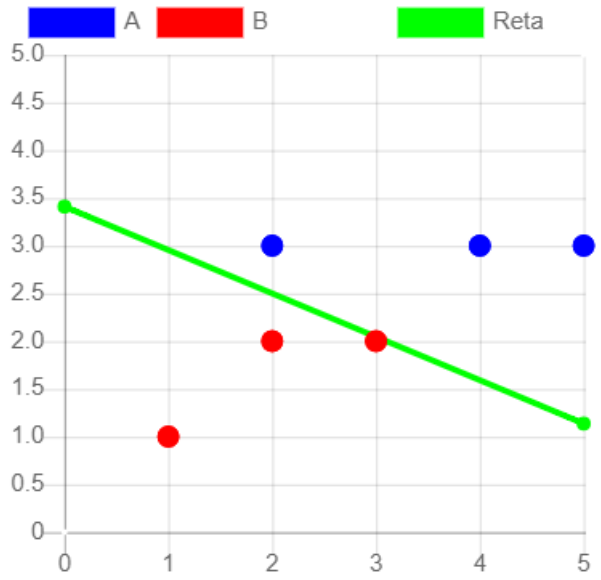
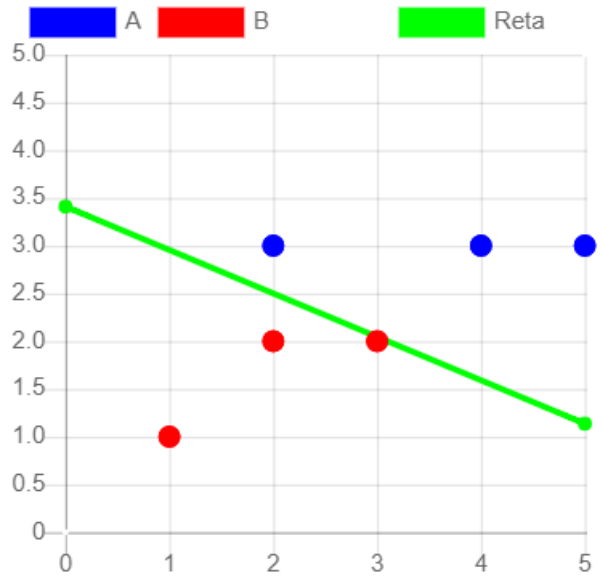


Figura 15 – Estado final do perceptron no passo $n = 49$. Com os valores dos pesos obtidos, o perceptron consegue agora classificar corretamente todas as amostras. O erro global é igual a zero

Simulador de Perceptron Simples									
Parâmetros			η	0.01	Passo	49		Limite	100
Pesos			w_1	0.133	w_2	0.293	w_0		-1
No.	x_1	x_2	d	f	o	e	e^2	Δ_1	Δ_2
1	4	3	0.41	1	1	0.00	0.00	0.00	0.00
2	1	1	-0.57	-1	-1	0.00	0.00	0.00	0.00
3	2	2	-0.15	-1	-1	0.00	0.00	0.00	0.00
4	2	3	0.15	1	1	0.00	0.00	0.00	0.00
5	5	3	0.55	1	1	0.00	0.00	0.00	0.00
6	3	2	-0.01	-1	-1	0.00	0.00	0.00	0.00
Tempo(s)			10 ▾	$\Sigma / \overline{\Delta}$			0	0.00	0.00

Fonte: elaborado com base em Frontino ([S.d.]).

Figura 16 – Representação no plano cartesiano do estado $n = 49$ do perceptron

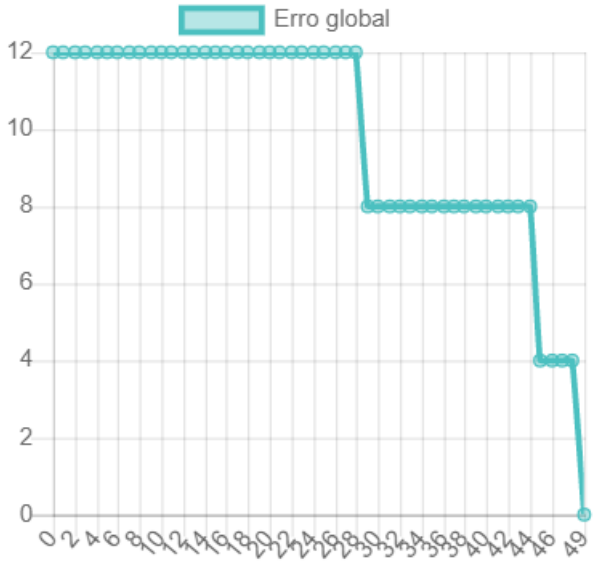


Nem sempre um perceptron consegue classificar corretamente um dado conjunto de amostras. Para funcionar corretamente, o perceptron deve trabalhar com classes que sejam **separáveis linearmente** (o caso do exemplo). Os padrões ou amostras pertencentes a uma classe devem estar suficientemente separados da outra classe para uma correta classificação.

Ao longo do treinamento do perceptron, pôde-se observar como o erro global E evoluiu (Figura 17). As correções introduzidas pelos deltas ajustaram os pesos, de maneira a minimizar o erro global ao longo dos passos (também chamado de *épocas*) do treinamento. Como o conjunto das amostras é separável linearmente, o erro global finalizou em zero, fazendo com que a rede classifique corretamente cada amostra em sua classe respectiva. Em casos mais complexos, o erro global é uma das únicas variáveis globais que permitirão indicar se o treinamento está sendo efetivo ou não. E, geralmente, se o conjunto não é separável linearmente, a tendência é o erro ser minimizado até certo valor.

A forma como o conjunto de treinamento é tratada no algoritmo de aprendizagem também pode variar. A atualização dos pesos pode acontecer **por amostra**, ou seja, a cada amostra calculada os pesos são atualizados; ou **por lote**, em que se calcula a variação média de todo o conjunto e depois se aplica a atualização dos neurônios (como no caso do perceptron apresentado anteriormente).

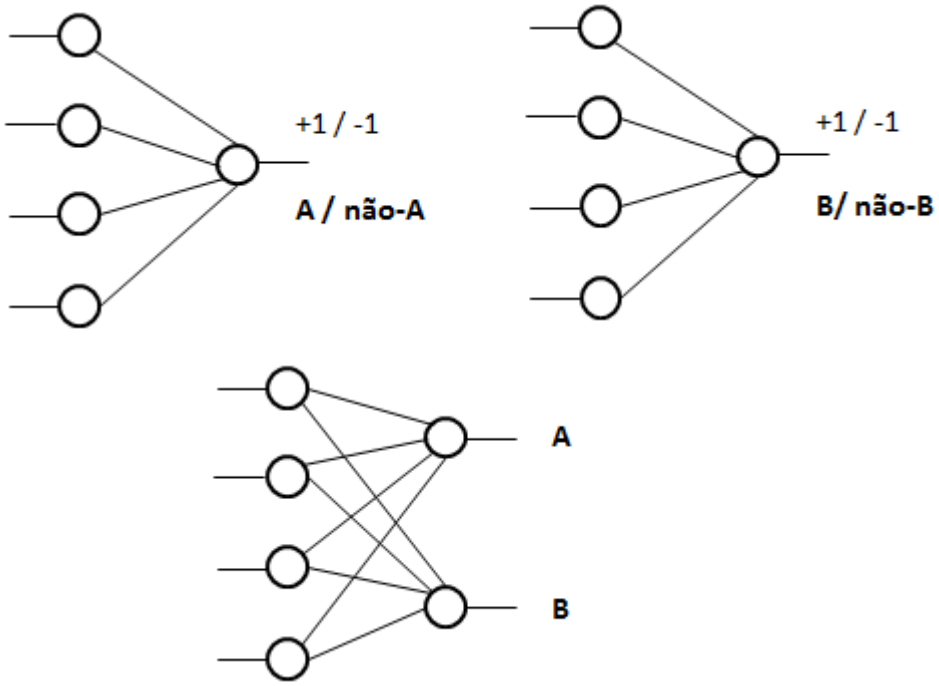
Figura 17 – Evolução do erro global no treinamento do perceptron



O exemplo da classificação dos parafusos permitiu visualizar como podemos aplicar um perceptron simples para somente duas classes. Em casos em que o problema exigir mais de duas classes, a arquitetura do perceptron deve ser alterada. Uma maneira é adaptar um perceptron para cada classe em que precisa ser treinado um conjunto de amostras. Por exemplo, ao invés de utilizar um perceptron para o problema de classificação dos parafusos definindo a saída +1 para a classe “A” e a saída -1 para a classe “B”, podemos utilizar dois perceptrons: o primeiro perceptron pode ser treinado para que a saída +1 seja atribuída à classe “A” e a saída -1 para que seja “não-A”. Da mesma forma, o segundo perceptron pode ser treinado para que a saída +1 seja para a classe “B”, e a saída –1 atribuído a “não-B” (Figura 18).

Generalizando, para N classes, podemos expandir essa mesma arquitetura, tendo então N perceptrons para fazer a classificação. O cuidado é fazer com que na fase de treinamento, uma amostra que deve ser classificada como “A” deve receber o valor +1 no primeiro perceptron, e no segundo perceptron esteja colocado o valor -1, ou seja, “não-B”, e vice-versa.

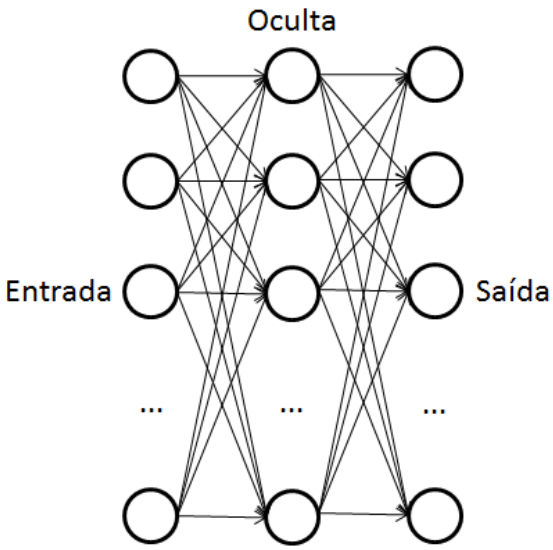
Figura 18 – Arquitetura diferenciada do perceptron simples, que pode ser generalizada para N classes



Em casos reais, nos quais existem vários atributos e uma quantidade maior de classes, deve-se utilizar o **perceptron multicamada.** Este se diferencia do perceptron simples por incluir camadas escondidas ou ocultas na RNA. Com a inclusão de camadas ocultas, o número de pesos ou sinapses aumenta consideravelmente (Figura 19). A consequência disso é a possibilidade de melhorar o mapeamento das entradas com as saídas. Um perceptron simples trabalha de forma linear, enquanto o perceptron multicamada tem condições de lidar com conjuntos de treinamento cuja separabilidade seja **não linear**.

A arquitetura com camadas ocultas requer algoritmos de aprendizagem que contemplem a atualização dos pesos relacionados às camadas internas. O processo de ativação acontece primeiramente nas camadas ocultas para depois chegar até a camada de saída. A retroalimentação do erro também é feita nos pesos que conectam a(s) camada(s) oculta(s).

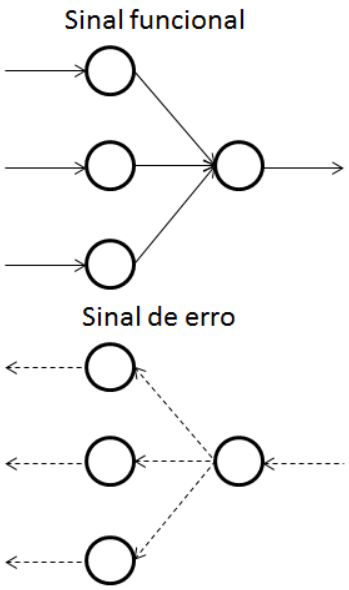
Figura 19 – Representação de uma rede perceptron multicamada (MLP), com uma camada oculta, com todos os nós conectados



A dinâmica do perceptron multicamada envolve o processamento de dois tipos de sinais (Figura 20) se propagando pela rede (Haykin, 2001, p. 186-187):

- **Sinal funcional**: é o sinal apresentado à camada de entrada referente aos atributos do vetor de amostras, que se propaga para a frente na rede, nó por nó, ativando os neurônios até a camada de saída.
- **Sinal de erro**: tem origem em um neurônio da camada de saída, porém, se propagando para trás na rede, ajustando os valores dos pesos ou sinapses.

Figura 20 – Representação do sinal funcional, que se propaga à frente, e do sinal de erro, que se retropropaga na rede



O algoritmo de retropropagação para o perceptron multicamada envolve o processo chamado de **descida de gradiente**. Esse processo busca calcular o gradiente local do erro (a direção para onde tende a crescer o valor do erro médio calculado), utilizando-o para corrigir os pesos sinápticos na direção contrária a esse gradiente, em busca do erro mínimo local.

TEMA 5 – APRENDIZADO PROFUNDO (*DEEP LEARNING*)

Deep learning tem sido uma das áreas mais estudadas na área de redes neurais artificiais, por se constituir num paradigma de modelagem de redes que leva em consideração outros aspectos além da conectividade da rede em si, com elementos na arquitetura da rede que a aproximam, inclusive, do paradigma biológico de captura e processamento de imagens no sistema visual humano. (Kriegeskorte; Golan, 2019).

No caso do perceptron multicamada, um dos problemas com relação ao aprendizado desse tipo de rede está relacionado com a representação do conhecimento feita nos seus pesos. Após o treinamento, a rede e seus pesos se manifestam como uma “caixa preta”, sendo impossível interpretar os valores dos pesos em termos de uma representação inteligível para o ser humano.

O aprendizado profundo resolve o problema central no aprendizado de representação, introduzindo representações que são expressas em termos de outras representações mais simples. O aprendizado profundo permite que o computador construa conceitos complexos a partir de conceitos mais simples.

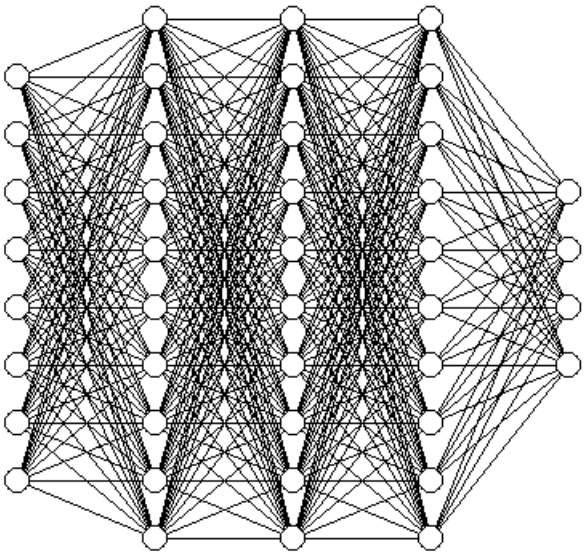
Para alcançar o seu objetivo, o *deep learning* abrange o estudo de uma série de técnicas e algoritmos de RNA, aprofundando conceitos de redes neurais alimentadas à frente (*feed forward*), camadas de entrada, ocultas e de saída, função custo, envolvendo o uso de outros tipos de redes, tais como redes convolutivas (*Convolutional Neural Networks* – CNN), redes neurais recorrentes (*Recurrent Neural Networks* – RNN), descida estocástica de gradiente (*Stochastic Gradient Descent*). Tais técnicas estão também aliadas ao uso de hardware contendo unidades de processamento gráfico (*Graphical Processing Units* – GPU) e computação baseada em GPU como tecnologia central para possibilitar o *deep learning* (Ketkar, 2017, p. 1-3; Medeiros, 2018, p. 158).

Na área de reconhecimento de padrões, como OCR (reconhecimento ótico de caracteres), processamento de sinais ou imagens, podem ser utilizadas redes neurais artificiais com camadas ocultas, de maneira a se fazer o reconhecimento

de um conjunto significativo de amostras. Por exemplo, em OCR, o banco de dados de caracteres escritos à mão MNIST contém 60 mil amostras de treinamento e 10 mil amostras de teste (Lecun et al., 1998). As imagens dos caracteres estão dispostas em matrizes de pixels 28x28.

Uma rede neural na forma de um perceptron multicamada pode ser imaginada para executar uma tarefa de reconhecimento ótico de caracteres. Os caracteres do banco de dados MNIST, com um tamanho de 28x28, totalizam 784 pixels. Pode-se elaborar um perceptron multicamada, tal como o da Figura 22, de 784 entradas e 10 saídas, contendo camadas ocultas para melhoramento da sua aprendizagem. Dessa forma, este processo “tabula rasa” se baseia inteiramente no aprendizado dos pesos, que começarão com valores aleatórios, para alcançar um erro global mínimo, no qual o processo de classificação dos dígitos manuscritos seja executado, com uma certa margem de erro.

Figura 21 – Representação de uma rede neural artificial com várias camadas ocultas



Com base na abordagem de aprendizagem profunda, pode-se inferir a arquitetura de uma rede que contenha elementos que explorem a característica espacial das imagens. Os caracteres podem sofrer translações (deslocamentos para os lados, para cima ou para baixo) ou rotações (sentido anti-horário ou horário) entre as amostras. O ponto é dotar a rede de elementos que possam executar o reconhecimento com **invariância** a rotações ou translações.

Um tipo de rede neural capaz de lidar com características de invariância e tirar vantagem da estrutura espacial dos caracteres é denominada de **rede neural convolucional** (*convolutional neural networks*). A **convolução** é uma operação matemática sobre duas funções que produz como resultado outra função que demonstra como a forma de uma modifica a outra. No contexto digital, é utilizada principalmente em filtragem de sinais (Mneney, 2008, p. 40; Tan; Jiang, 2013, p. 72).

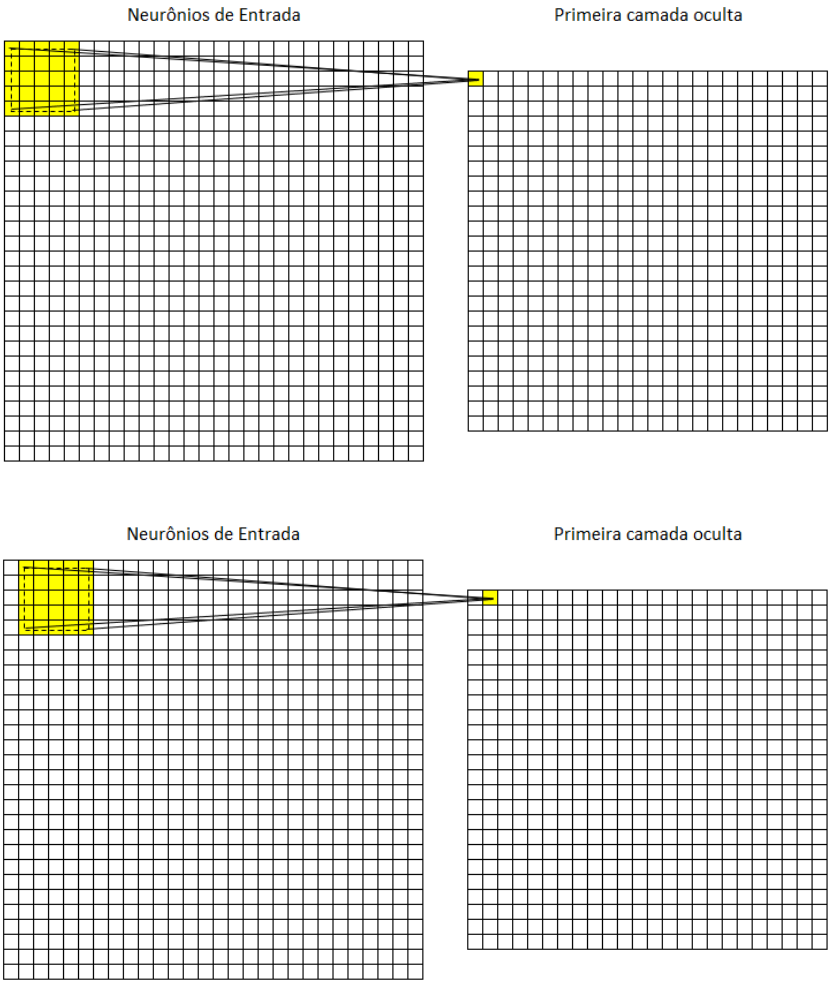
As redes convolucionais utilizam uma arquitetura especial que é particularmente bem adaptada para classificar imagens. O uso dessa arquitetura torna mais rápido o treinamento das redes convolucionais. Isso, por sua vez, nos ajuda a treinar redes profundas de várias camadas, que são muito boas na classificação de imagens. Hoje, redes convolucionais profundas ou alguma variante próxima são usadas na maioria das redes neurais para reconhecimento de imagens. As redes neurais convolucionais usam três ideias básicas: **campos receptivos locais**, **pesos compartilhados** e **camadas de aglutinação** (Nielsen, 2019).

Campos receptivos locais: nas camadas totalmente conectadas mostradas em um perceptron multicamada, as entradas foram representadas como uma linha vertical de neurônios. Em uma rede convolucional, as entradas podem ser visualizadas como um quadrado de 28 × 28 de neurônios, cujos valores correspondem às intensidades de 28 × 28 pixels usados como entradas.

Na conexão dos pixels de entrada a uma camada de neurônios ocultos, não serão conectados todos os pixels de entrada a todos os neurônios ocultos. Em vez disso, faz-se apenas conexões em pequenas regiões localizadas da imagem de entrada. Para ser mais preciso, cada neurônio na primeira camada oculta será conectado a uma pequena região dos neurônios de entrada, por exemplo, uma região 5 x 5, correspondente a 25 pixels de entrada (Figura 23).

Essa região na imagem de entrada é chamada de **campo receptivo local** para o neurônio oculto. É uma pequena janela nos pixels de entrada. Cada conexão aprende um peso. Esse neurônio oculto em particular pode ser pensado como aprendendo a analisar seu campo receptivo local específico. Em seguida, desliza-se o campo receptivo local por toda a imagem de entrada. Para cada campo receptivo local, existe um neurônio oculto diferente na primeira camada oculta.

Figura 22 – Camada posterior da rede com 24 x 24 neurônios, combinados a partir de campos receptivos locais 5 x 5 da entrada da rede 28 x 28



Fonte: adaptado de Nielsen, 2019, p. 171.

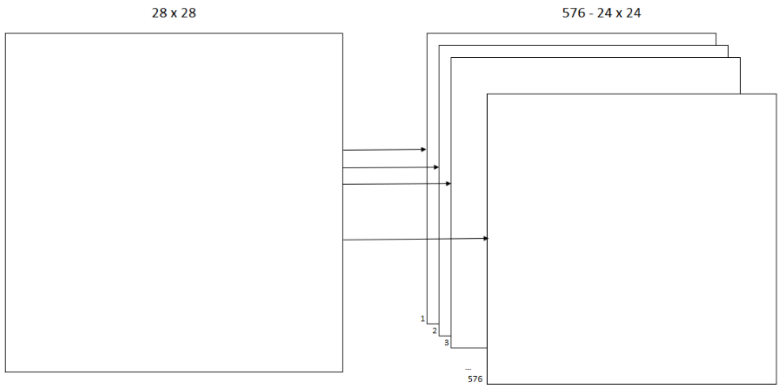
E assim, sucessivamente, constrói-se a primeira camada oculta. Deve-se notar que, se tivermos uma imagem de entrada 28 x 28 e campos receptivos locais 5 x 5, haverá 24 x 24 neurônios na camada oculta. Isso ocorre porque só é possível deslocar o campo receptivo local 23 neurônios para o lado (ou 23 neurônios para baixo) antes de colidir com o lado direito (ou inferior) da imagem de entrada.

Compartilhamento de pesos: é onde acontece o processo de convolução propriamente dito, também denominada de **camada convolucional**. Para o exemplo do campo receptivo local 5 x 5, todos os 25 pesos serão os mesmos, por isso que se afirma que os pesos são compartilhados. Isso significa que todos os neurônios da primeira camada oculta vão detectar exatamente o mesmo recurso apenas em locais diferentes na imagem de entrada. Para entender por que isso

faz sentido, suponha que os pesos e o viés sejam tais que o neurônio oculto possa identificar, digamos, uma borda vertical em um campo receptivo local específico. É provável que essa capacidade também seja útil em outros locais da imagem (Figura 24). E, portanto, é útil aplicar o mesmo detector de recursos em qualquer lugar da imagem. Esta característica permite que as redes convolucionais lidem com a característica de **invariância** de imagens: caso a imagem sofra uma translação, ou uma pequena rotação, isso não afetará o processo de detecção.

Por esse motivo, às vezes denomina-se o mapa da camada de entrada para a camada oculta de **mapa de características** (*feature map*). Dessa forma, os pesos que definem o recurso de mapeamento de **pesos compartilhados**. Diz-se também que os pesos compartilhados definem um **kernel** ou **filtro** (Nielsen, 2019, p. 172).

Figura 23 – A camada convolucional equivale a várias camadas produzidas a partir dos campos receptivos locais. O número de 576 foi obtido a partir do descolamento da máscara 5 x 5 sobre o campo de entrada. Entretanto, valores arbitrários podem ser definidos, de acordo com a arquitetura desejada



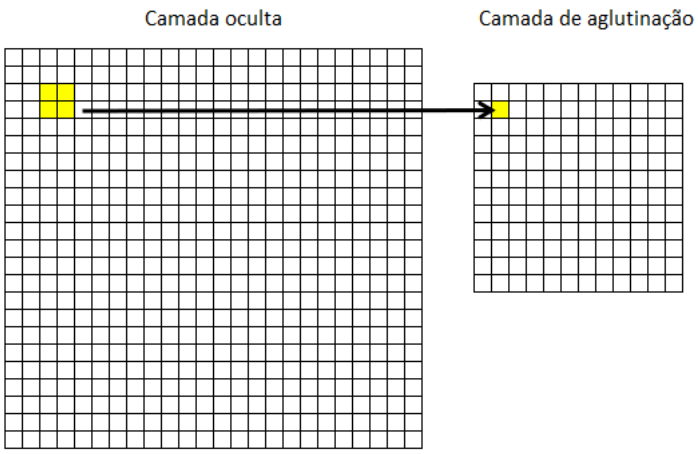
Fonte: adaptado de Nielsen, 2019, p. 172.

Camada de aglutinação (*pooling layer*): além das camadas de compartilhamento de peso descritas acima, as redes neurais convolucionais também contêm camadas de aglutinação. As camadas de aglutinação geralmente são usadas imediatamente ao final da rede convolucional. O que essas camadas fazem é simplificar as informações na saída da camada anterior (Figura 25).

Em suma, uma camada de aglutinação pega cada saída do mapa de recursos da camada convolucional e prepara um mapa condensado. Por exemplo, cada unidade na camada de pool pode resumir uma região de (por exemplo) 2 x 2 neurônios na camada anterior. Pode-se elaborar uma camada de aglutinação

com apenas uma unidade de aglutinação que gera a ativação máxima na região de entrada 2 x 2.

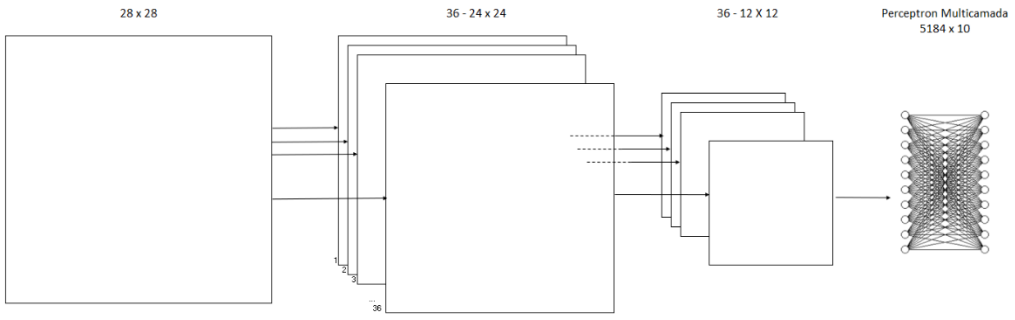
Figura 24 – Ilustração mostrando a formação de uma camada de aglutinação a partir de um campo 2 x 2 da camada oculta



. Fonte: adaptado de Nielsen, 2019, p. 172.

Ao final, pode-se utilizar a camada de aglutinação como entrada para uma rede perceptron multicamada. Diferente das camadas anteriores, nas quais os valores dos pesos serão predefinidos, os pesos dessa rede poderão ser treinados para o conjunto de caracteres que fizer parte da amostra. A figura 26 mostra, por fim, a arquitetura da rede para o reconhecimento de caracteres.

Figura 25 – Arquitetura da rede para o reconhecimento de caracteres



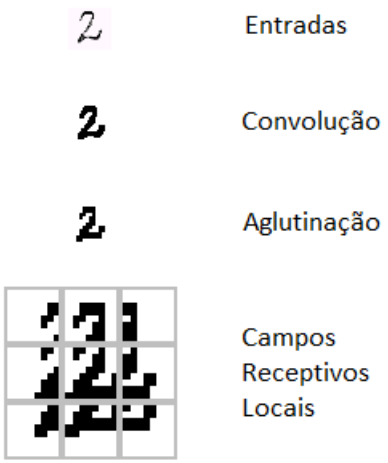
Fonte: adaptado de Nielsen, 2019, p. 175.

A ideia central com esse exemplo foi mostrar que, à luz do paradigma do aprendizado profundo, uma rede neural pode ser adaptada de maneira a incorporar características do problema. Ao invés de se conectar 28 x 28 entradas, com várias camadas ocultas, aos neurônios de saída, buscando-se treinar todos os pesos, utiliza-se uma característica espacial do problema de forma a considerar

pesos predefinidos para as primeiras camadas, ficando apenas a última para ser treinada. Quanto às questões de performance da rede, também são indiscutíveis. A rede híbrida obtida no exemplo precisará treinar apenas 51.840 pesos, ao invés de 784 (28 x 28 entradas) e supondo 3 camadas de 784 neurônios e uma camada de saída de 10 neurônios: 784 x 784 x 784 x 784 x 10 = 3.778.019.983.360 pesos!

Outro exemplo de reconhecimento de dígitos que permite verificar o efeito sobre a imagem do caractere pode ser visto na figura 27 (Medeiros, 2006, p. 48). Nesta implementação, a rede foi construída de forma a permitir primeiro o compartilhamento dos pesos (convolução) de dígitos em imagens de entrada de 32 x 32; em seguida, é feita a camada de aglutinação para, por último, produzir 9 campos receptivos locais.

Figura 26 – Efeitos das operações das camadas em uma rede convolucional



Fonte: adaptado de Medeiros, 2016, p. 48.

REFERÊNCIAS

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. Deep Learning. **MIT Press**, 2016. Disponível em: <http://www.deeplearningbook.org/>. Acesso em: 7 abr. 2020.

HAYKIN, S. **Redes neurais**: princípios e prática. Porto Alegre: Bookman, 2001.

KETKAR, N. **Deep Learning with Python**: a Hands-on Introduction. California: Apress Media, 2017.

KRIEGESKORTE, N.; GOLAN, T. Neural network models and deep learning – a primer for biologists. **Columbia University**, 2019. Disponível em: <https://arxiv.org/ftp/arxiv/papers/1902/1902.04704.pdf>. Acesso em: 7 abr. 2020.

LECUN, Y. et al. Gradient-Based Learning Applied to Document Recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 2278-2324, nov. 1998.

MEDEIROS, L. F. **Inteligência artificial aplicada**: uma abordagem introdutória. Curitiba: Intersaberes, 2018.

MEDEIROS, L. F. **Redes neurais em Delphi**. 2. ed. Florianópolis: Visualbooks, 2006.

MNENEY, S. H. **An Introduction to Digital Signal Processing**: a Focus on Implementation. Aalborg, Denmark: Rivers Publishing, 2008.

NIELSEN, M. **Neu**ral Networks and Deep Learning. **Nielsen**, 2019. Disponível em: <http://neuralnetworksanddeeplearning.com/>. Acesso em: 7 abr. 2020.

PROFESSOR FRONTINO. Simulador de Perceptron Simples. **Professor Frontino**, [S.d]. Disponível em: <http://professorfrontino.com.br/ia/simulator.html>. Acesso em: 7 abr. 2020.

ROSENBLATT, F. The Perceptron: A Probabilistic Model Information Storage and Organization in the Brain. **Psycological Review**, v. 65, n. 6, 1958. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf>. Acesso em: 7 abr. 2020.

ROSENBLATT'S Contributions. **Pace University**, [S.d.]. Disponível em: <http://csis.pace.edu/~ctappert/srd2011/rosenblatt-contributions.htm>. Acesso em: 7 abr. 2020.

TAN, L.; JIANG, J. **Digital Signal Processing**: Fundamentals and Applications. 2. ed. Oxford: Academic Press, 2013.