Victor Rodrigues de Oliveira

Descobrindo Redes Neurais Artificiais - Minicurso sobre os modelos: Perceptron, Adaline e Perceptron Multicamadas

Dissertação de Mestrado apresentada ao Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, para obtenção do título de Mestre em Matemática.

Instituto Federal de Educação Ciência e Tecnologia de São Paulo

Campus São Paulo

Programa de Mestrado Profissional em Matemática em Rede Nacional - PROFMAT

Orientador: Prof. Dr. Marco Aurélio Granero Santos

São Paulo - SP

2025

Lista de ilustrações

Lista de tabelas

Lista de Algoritmos

Sumário

1 Perceptron Multicamadas

A composição de Perceptrons, que forma uma RNA chamada de Perceptron Multicamadas (PMC), contorna limitações que o Perceptron unitário possui em relação às condições que precisam ser satisfeitas para que ele possa ser utilizado eficientemente e às possibilidades de separação de dados que pode realizar (Bottou, 2017; Minsky; Papert, 1988).

Quando é estabelecida uma conexão direta entre a saída de um neurônio com a entrada de outro, criam-se camadas. Ao se utilizar redes neurais de várias camadas são introduzidos novos algoritmos de treinamento, cada um com seus respectivos potenciais e limitações.

Na Figura 1 é apresentado um exemplo de rede PMC. As linhas indicam as direções que os dados seguem, podendo alternar se os sentidos, da esquerda para a direita, ou o contrário, de acordo com a etapa do treinamento em questão. Os círculos são os neurônios e já na primeira camada, em vermelho, cada um dos três neurônios estão recebendo todas as variáveis x_1, x_2 e x_3 da amostra em questão.

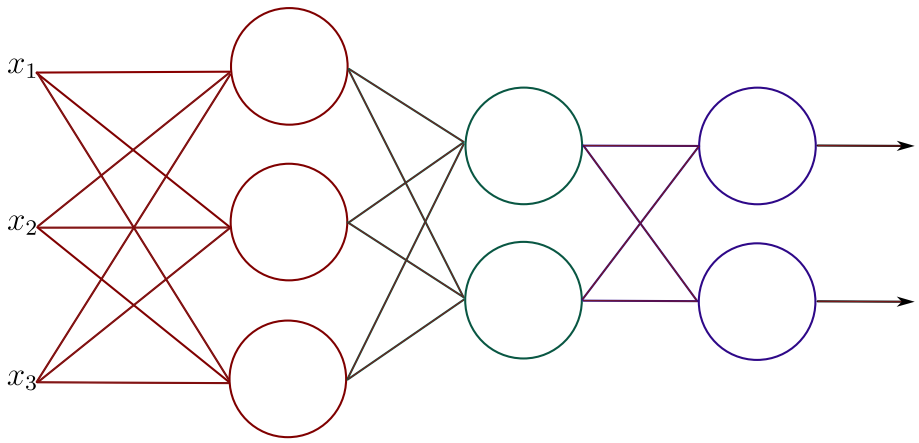


Figura 1 – Rede Perceptron Multicamadas com três variáveis, três camadas e duas saídas. Elaborada pelo autor.

Continuando com o exemplo da Figura 1, os neurônios em vermelho formam a primeira camada da RNA ou camada de entrada. As saídas dos neurônios desta camada estão diretamente ligadas com as entradas de cada um dos neurônios em verde. Isto significa que são as saídas dos neurônios da primeira camada que serão combinadas linearmente com os pesos sinápticos de cada um dos dois neurônios em verde, neurônios da segunda camada. Consequentemente, para a terceira camada, neurônios em azul, as saídas da segunda camada servirão de entradas.

A primeira camada do exemplo apresentado na Figura 1 é a camada de entrada e,

junto com a segunda camada, são conhecidas como camadas ocultas. Estas são designadas assim pois o operador/treinador da RNA não terá acesso a saída delas. A terceira camada é chamada de camada de saída e são os neurônios destas camadas que apresentarão as saídas para os dados fornecidos à RNA.

É importante evidenciar que, não só no exemplo da Figura 1, mas em todas as RNA do tipo PMC, as saídas dos neurônios serão suas saídas das funções de ativação. Outro fato importante é que a função de ativação escolhida é a adotada por todos os neurônios da rede.

Deste modo, todos os neurônios da segunda camada em diante recebem os dados da camada anterior, as ponderam com seus respectivos pesos sinápticos, comparam com os respectivos limiares e então fornecem esta combinação linear para as funções de ativação, enquanto os neurônios da primeira camada realizam estes mesmos processos mas com as variáveis dos dados das entradas.

Os pesos sinápticos de cada neurônio são independentes dos outros neurônios. Isto significa que eles devem ser treinados e devidamente responsabilizados quando a rede apresentar um erro. O algoritmo de treinamento que será estudado neste trabalho é o *backpropagation*.

Este algoritmo foi escolhido para ser estudado neste capítulo devido aos avanços que pode proporcionar em relação a outros algoritmos que existiam na época em que foi apresentado, inspirando o desenvolvimento de adaptações e melhorias para casos específicos (Widrow; Lehr, 1990).

Antes de apresentar o treinamento de uma rede neural em maiores detalhes, será implementada uma função booleana que não poderia ser implementada com o Perceptron de camada única ou com o Adaline, a função booleana ou-exclusivo ("XOU").

1.1 Implementando a função booleana ou-exclusivo "XOU"

Como visto na Figura **??**, a função booleana "XOU" não é linearmente separável e portanto, não existe um modo de somente um neurônio ser capaz de implementá-la. Para que seja possível a implementação desta função, será criada uma rede neural com camadas.

Antes de definir a quantidade de neurônios e de camadas, deve-se preparar a implementação para que possa ser feita segundo os moldes do que foi realizado no capítulo **??** para as funções booleanas "OU" e "E".

Primeiramente constrói-se a tabela verdade da função "XOU":

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Tabela 1 – Tabela verdade da função booleana "XOU".

onde o sinal de $\oplus$ indica a atuação da função booleana "XOU" nas variáveis x_1 e x_2.

A arquitetura da RNA do tipo PMC que será adotada foi baseada na que foi apresentada por Kovács (2023) que utiliza a decomposição da função "XOU" com o uso de três funções booleanas. A função "XOU" pode ser expressa como combinação linear das funções "E", a função "NEGAÇÃO"($\neg$) e a função "OU".

A função booleana "NEGAÇÃO" inverte os valores binários que recebe, ou seja, se a entrada for "1" ela apresenta "0" como saída e vice-versa. A tabela verdade da função "NEGAÇÃO" para as variáveis de entrada x_1 e x_2 pode ser vista na Tabela 2.

x_1	x_2	$\neg x_1$	$\neg x_2$
0	0	1	1
0	1	1	0
1	0	0	1
1	1	0	0

Tabela 2 – Tabela verdade da função "NEGAÇÃO". Elaborada pelo autor.

Recordando a notação adotada para as funções "E" e "OU", pode-se fazer, ao considerar como f a função booleana "XOU",

$$\begin{aligned} f(x_1, x_2) &= x_1 \oplus x_2 \\ &= (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2), \end{aligned}$$

e, fazendo $f_1 = x_1.(\neg x_2)$ e $f_2 = (\neg x_1).x_2$, tem-se $f = f_1 \vee f_2$.

É possível ver a equivalência do uso das funções para representar a função "XOU", com a saída desta função, na Tabela 3.

x_1	x_2	$\neg x_1$	$\neg x_2$	$x_1 \wedge \neg x_2$	$\neg x_1 \wedge x_2$	$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$	$x_1 \oplus x_2$
0	0	1	1	0	0	0	0
0	1	1	0	0	1	1	1
1	0	0	1	1	0	1	1
1	1	0	0	0	0	0	0

Tabela 3 – Tabela-verdade da função "XOU" $(A \oplus B)$ através de sua decomposição usando "NEGAÇÃO", "E" e "OU". Elaborada pelo autor.

Com isto, para esta RNA serão representados três neurônios: N_1, N_2 e N_3, associados às funções f_1, f_2 e f, respectivamente, conforme ilustrado na Figura 2.

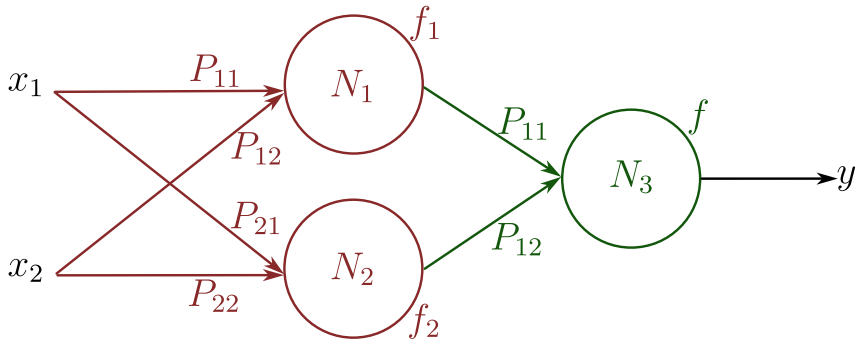


Figura 2 – Arquitetura envolvida na implementação da função booleana "XOU" com três neurônios de McCulloch-Pitts. Elaborado pelo autor.

De acordo com a decomposição apresentada, na Figura 2, a arquitetura desta RNA terá dois neurônios na primeira camada, responsáveis por receberem as entradas x_1 e x_2 e um neurônio na camada de saída que receberá as saídas dos neurônios da primeira camada e fornecerá a saída y da rede.

Na tabela verdade apresentada na Tabela 1, percebe-se que existem somente dois casos em que as saídas "1" são resultados da função "XOU". Com isso, pode-se atribuir cada um dos dois casos, respectivamente aos neurônios N_1 e N_2 e, em seguida, utiliza-se o neurônio N_3 para unir essas duas saídas em uma só.

O neurônio N_1, responsável por resolver $f_1(x_1, x_2) = x_1.(\neg x_2)$, encontra, por exemplo, uma solução tomando $P_{11}^{(1)} = \dfrac{1}{2}$ e $P_{12}^{(1)} = -\dfrac{1}{2}$, conforme ilustrado na Tabela 4.

(x_1, x_2)	$P_{11}^{(1)}.x_1 + P_{12}^{(1)}.x_2 - \theta = u_1$
$(0,0)$	$\frac{1}{2}.0 - \frac{1}{2}.0 = \quad 0 - \theta$
$(0,1)$	$\frac{1}{2}.0 - \frac{1}{2}.1 = -\frac{1}{2} - \theta$
$(1,0)$	$\frac{1}{2}.1 - \frac{1}{2}.0 = \quad \frac{1}{2} - \theta$
$(1,1)$	$\frac{1}{2}.1 - \frac{1}{2}.1 = \quad 0 - \theta$

Tabela 4 – Combinação linear das variáveis com os pesos do primeiro neurônio da camada entrada. Elaborada pelo autor.

Com isso, o funcionamento do neurônio N_1 pode ser descrito pela função de ativação degrau definida pela Equação (1.1).

$$f_1(x_1, x_2) = \begin{cases} 1, \ se \ u_1 \geqslant 0 \\ 0, \ se \ u_1 < 0. \end{cases} \tag{1.1}$$

Os coeficientes fornecidos para o neurônio N_1, juntamente com um limiar $\theta = \frac{1}{4}$ que foi estipulado de acordo com a função degrau apresentada na Equação (**??**), podem

ser utilizados como coeficientes de uma reta que separa linearmente os dados, como pode ser visto na Figura 3.

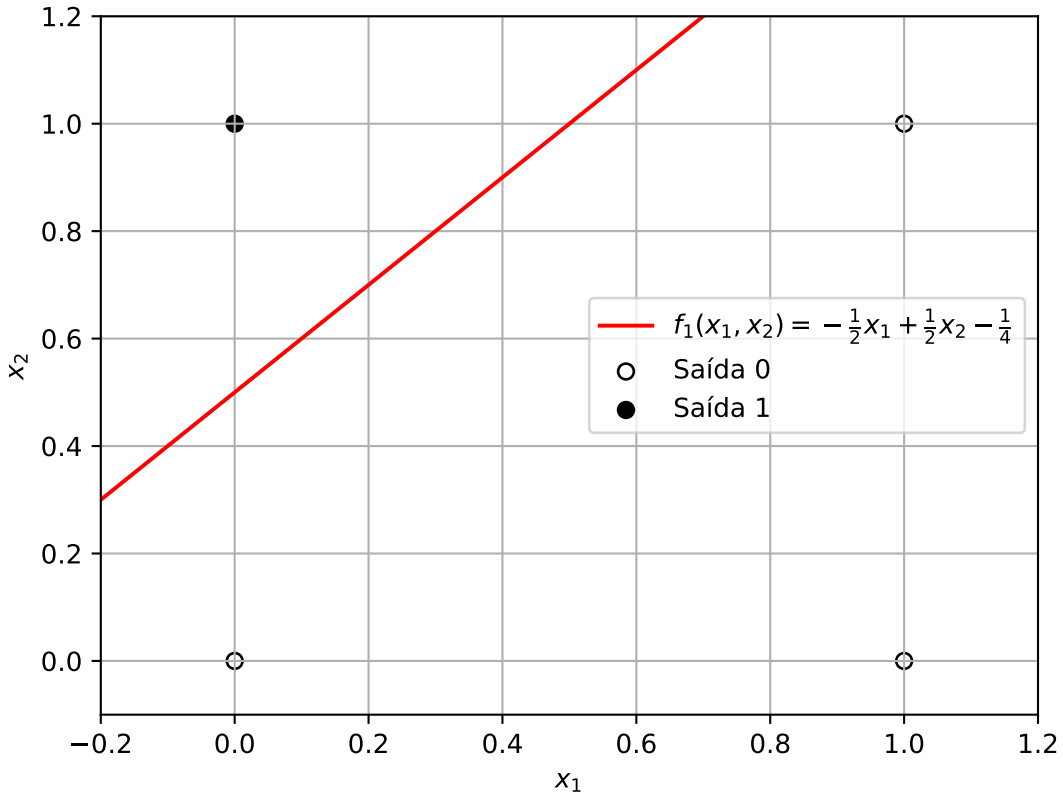


Figura 3 – Reta obtida com os pesos e limiar selecionados para f_1. Elaborado pelo autor.

De modo análogo, para N_2 com $f_2(x_1, x_2) = (\neg x_1).x_2$, pode-se fazer $P_{21}^{(1)} = -\frac{1}{2}$ e $P_{22}^{(1)} = \frac{1}{2}$ de modo a obter a Tabela 5:

(x_1, x_2)	$P_{21}^{(1)}.x_1 + P_{22}^{(1)}.x_2\theta = u_2$
$(0,0)$	$-\frac{1}{2}.0 + \frac{1}{2}.0 = 0 - \theta$
$(0,1)$	$-\frac{1}{2}.0 + \frac{1}{2}.1 = \frac{1}{2} - \theta$
$(1,0)$	$-\frac{1}{2}.1 + \frac{1}{2}.0 = -\frac{1}{2} - \theta$
$(1,1)$	$-\frac{1}{2}.1 + \frac{1}{2}.1 = 0 - \theta$

Tabela 5 – Combinação linear das variáveis com os pesos do segundo neurônio da camada de entrada. Elaborada pelo autor.

Utilizando a mesma função de ativação, definida em, (1.1), tem-se, para o neurônio

N_2:

$$f_2(x_1, x_2) = \begin{cases} 1, \ se \ u_2 \geqslant 0 \\ 0, \ se \ u_2 < 0 \end{cases} . \tag{1.2}$$

De modo análogo ao que foi feito para a função f_1, pode-se elaborar um gráfico ao utilizar os pesos sinápticos e limiar atribuído como $\theta = \frac{1}{4}$ como coeficientes de uma reta, como pode ser visto na Figura 4.

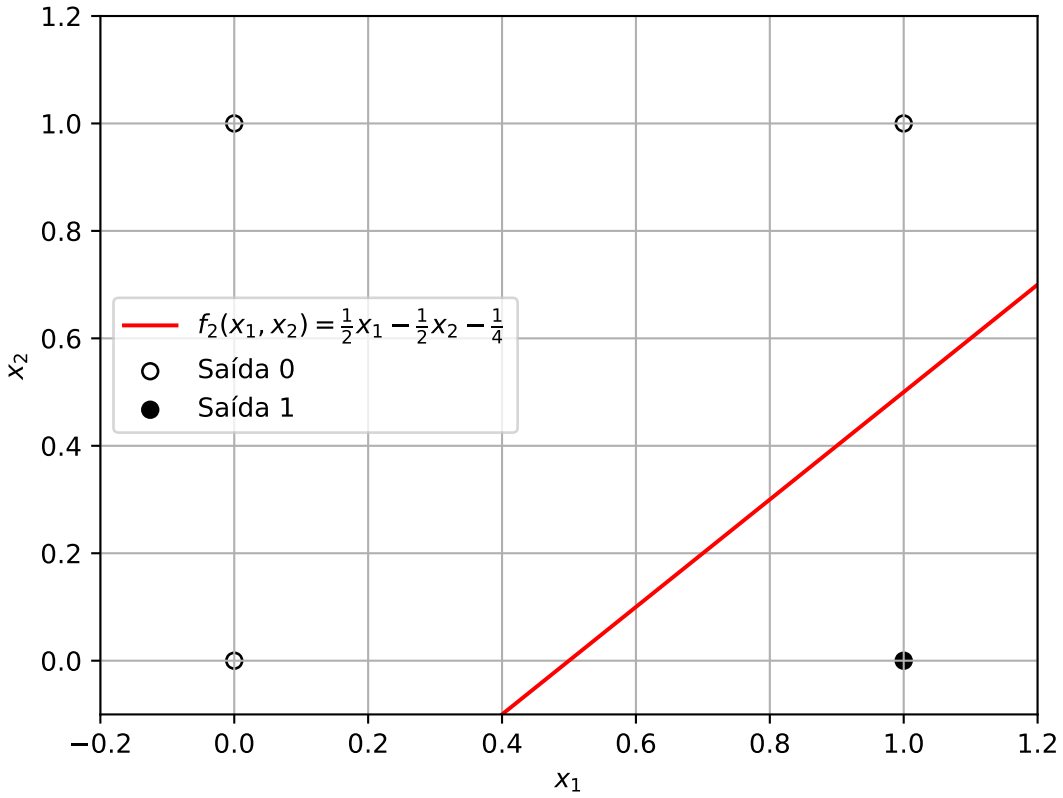


Figura 4 – Reta obtida com os pesos e limiar selecionados para f_2. Elaborado pelo autor.

Como os neurônios N_1 e N_2 pertencem a primeira camada de neurônios e N_3 à camada de saída, o neurônio N_3 não trabalhará diretamente com as entradas x_1 e x_2, mas sim receberá as respectivas saídas dos neurônios N_1 e N_2.

Estabelecidas as saídas dos neurônios da primeira camada, faz-se os ajustes necessários aos pesos da segunda camada de forma que a saída dela seja correspondente a da função que se deseja implementar.

Com base nas entradas x_1 e x_2 e nas saídas u_1 e u_2, dos neurônios da camada de entrada, é possível observar que o par ordenado $(N_1, N_2) = (1, 1)$ não fará parte do

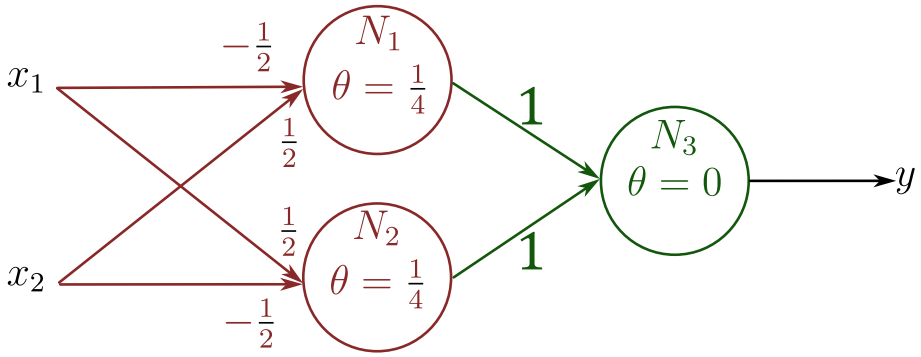


Figura 5 – Proposta de RNA para função "XOU" com pesos sinápticos atribuídos. Elaborado pelo autor.

domínio de f, pois independentemente de qualquer entrada atribuída à rede, a camada de entrada não apresentará simultaneamente o resultado mencionado. Tal fato pode ser observado na Tabela 6, adaptada da Tabela 3.

x_1	x_2	N_1	N_2
0	0	0	0
0	1	1	0
1	0	0	1
1	1	0	0

Tabela 6 – Tabela-verdade da função "XOU" $(A \oplus B)$ e sua decomposição usando "NEGAÇÃO", "E" e "OU". Elaborada pelo autor.

Dessa forma, pode-se adotar $P_{11}^{(2)} = P_{12}^{(2)} = 1$, como os pesos sinápticos de N_3, e construir a tabela de saída da rede como:

(N_1, N_2)	$P_{11}^{(2)} \cdot N_1 + P_{12}^{(2)} \cdot N_3 = u_3$
$(0, 0)$	$1.0 + 1.0 = 0$
$(0, 1)$	$1.0 + 1.1 = 1$
$(1, 0)$	$1.1 + 1.0 = 1$
$(0, 0)$	$1.0 + 1.0 = 0$

Tabela 7 – Dados da primeira camada fornecidos à camada de saída.

À saída de N_3 aplica-se a função de ativação como definida da Equação (**??**), com $\theta = 0$, de modo a obter:

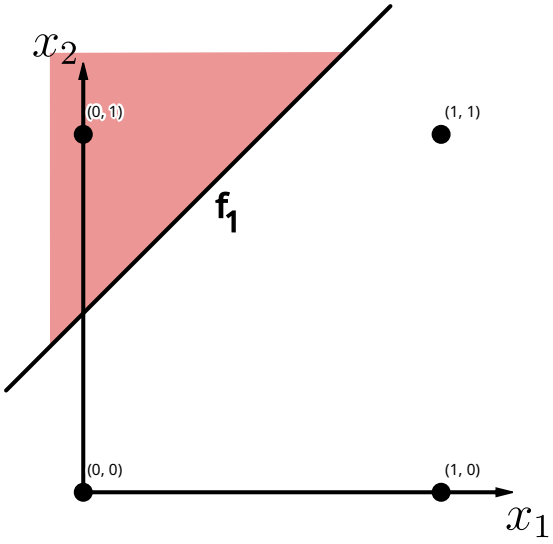
$$f(N_1, N_2) = \begin{cases} 1, \ se \ u_3 \geqslant 0 \\ 0, \ se \ u_3 < 0. \end{cases} \tag{1.3}$$

O diagrama na Figura 2, com os pesos atualizados, pode ser visto na Figura 5.

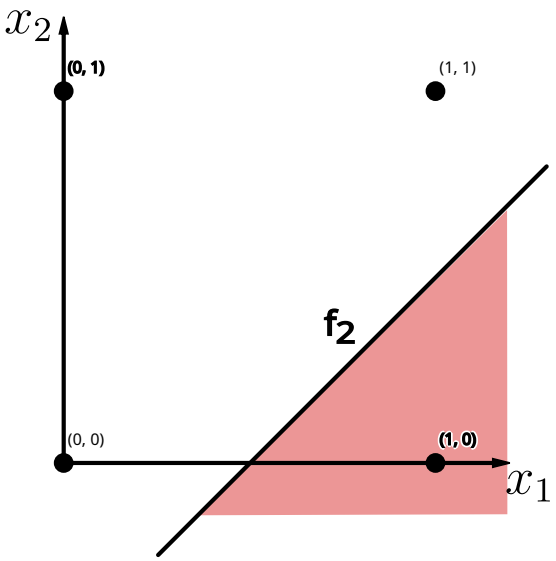
Com a atribuição dos pesos finalizada é possível escrever a saída da RNA em função dos dados das entradas, como pode ser visto na Equação (1.4).

$$f(x_1, x_2) = \begin{cases} 1, \ se \ x_1 \neq x_2 \\ 0, \ se \ x_1 = x_2. \end{cases} \tag{1.4}$$

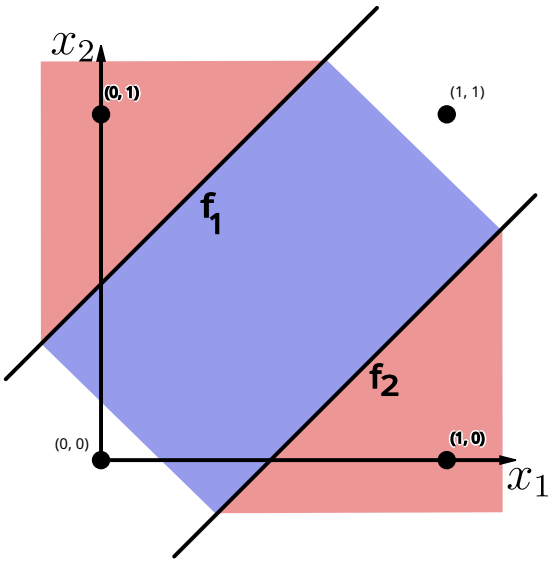
Uma representação acerca do resultado obtido por esta RNA é ilustrada na Figura 6, em que a Figura 6a apresenta isoladamente a representação geométrica da função f_1 e a Figura 6b a representação geométrica da função f_2.



(a) Função f_1.



(b) Função f_2.



(c) Função f.

Figura 6 – Composição de funções para a função booleana "XOU".
Elaborado pelo autor.

Na Figura 6c tem-se a representação geométrica da solução apresentada pela a

RNA ao problema do ou-exclusivo, onde a região em vermelho apresenta um dos resultados possíveis e a região em azul a outra possível solução.

Assim como foi feito no capítulo, **??**, a atribuição dos pesos sinápticos foi realizada manualmente e de modo conveniente. Para uma RNA com maior complexidade, mais neurônios, mais camadas e com maior volume de dados de amostras, a atribuição manual de pesos sinápticos e de seus ajustes se torna inviável e, portanto, efetuar o treinamento por um algoritmo se torna necessário.

No livro Perceptrons: An Introduction to Computational Geometry (1988), os autores, apesar de incentivarem o estudo do Perceptron, estendem para as RNA do tipo PMC as limitações que trazem em seu texto, apesar de não dissertarem muito a respeito.

Este livro, segundo Bottou (2017) e Silva, Spatti e Flauzino (2016), foi um fator importante para a queda de investimentos no desenvolvimento de pesquisas e tecnologias envolvendo Redes Neurais Artificiais, que ocorreu logo após seu lançamento. Porém, ainda de acordo com Bottou (2017), os incentivos foram direcionados ao estudo de processadores de sinais adaptativos, categoria a qual o Adaline se enquadra.

Segundo Widrow e Lehr (1990), uma RNA em que vários Adaline se conectam é denominada de Madaline[1] e, tais redes detém algumas funções booleanas fixas conectando alguns neurônios, além disso, diferentes algoritmos de treinamentos foram desenvolvidos para o Madaline, porém, o algoritmo Madaline Rule III se mostrou matematicamente equivalente ao treinamento efetuado em uma rede PMC com o uso do algoritmo *backpropagation*.

1.2 O treinamento do Perceptron Multicamadas

O treinamento do PMC também é do tipo supervisionado. Os erros calculados entre as saídas geradas pela rede e as saídas desejadas constituem o papel principal do ajuste dos pesos sinápticos e, assim como no Adaline, a tentativa de minimizar este erro estabelece o caminho a ser percorrido no treinamento de uma rede PMC.

Este treinamento pode ser efetuado em *lotes*, *mini-lote*s ou *em linha*. Estas três estratégias se referem ao modo como as amostras de treinamento são utilizadas.

Segundo Géron (2019), o treinamento em *lotes* é muito lento se comparado aos outros dois. Isto se deve ao grande volume de dados que a rede deve armazenar para executar as etapas do treinamento. Por outro lado, Haykin (2001) afirma que este é o método que tem maior estabilidade na convergência da rede para valores ótimos de pesos sinápticos.

O treinamento efetuado *em linha* atualiza os pesos sinápticos após o processamento

[1] Acrônimo para *Many Adaline*, em português, Muitos Adaline.

de cada amostra que é fornecida a rede. Este método é mais econômico em relação à custos computacionais como processamento e armazenamento, porém é instável. A cada amostra inserida, a convergência dos pesos sinápticos pode se afastar do mínimo global (Haykin, 2001).

Uma alternativa aos dois seria o uso de *mini-lotes* de dados de amostras. Esta é uma estratégia que fica no meio termo entre a estratégia de *lotes* e a *em linha* em todos os quesitos, pois utiliza pequenos lotes das amostras, sendo estes partes menores do que o lote completo.

Diferentemente do que foi feito nos capítulos anteriores, para o PMC será utilizado o treinamento *em linha*. Esta estratégia de treinamento é adotada para que as passagens realizadas aqui estejam mais próximas das utilizadas por Silva, Spatti e Flauzino (2016), Haykin (2001), e Géron (2019), por exemplo. Isto é feito com a expectativa de que, utilizar uma amostra por vez, possibilite um melhor entendimento do que está sendo feito, já que a notação utilizada pode fornecer maiores detalhes no processo de treinamento.

1.3 O algoritmo *backpropagation*

O *backpropagation*, também chamado de retropropagação, recebe seu nome pois, após avaliar os erros apresentados pelas saídas da rede em relação às saídas desejadas, retorna este erro juntamente com um fator de correção a todos os neurônios.

Este algoritmo foi apresentado no fim da década de 1980 e utiliza a ideia de controle e redução dos erros entre as saídas produzidas pela RNA e as respectivas saídas desejadas, similarmente ao que é feito no Adaline.

Todos os neurônios de um PMC estão conectados seja diretamente ou indiretamente, por meio de outros neurônios. Isto significa que um neurônio com pesos mal ajustados pode acarretar no mau funcionamento de toda RNA, afetando as saídas produzidas pela rede. Segundo Haykin (2001), o algoritmo *backpropagation* consegue atribuir níveis de responsabilidades aos neurônios que mais, ou menos, desviam a rede da saída desejada e ajusta seus pesos de acordo.

O algoritmo de treinamento *backpropagation* contém duas fases. A primeira consiste na passagem dos dados das amostras até a apresentação da saída da RNA, e a segunda fase, em que um erro é calculado e retornado até os neurônios da primeira camada. Estas fases se repetem, alternadamente, até que a rede determine que os pesos estão ajustados segundo um critério adotado para o encerramento do treinamento.

Segundo Widrow e Lehr (1990), a mudança da função degrau, empregada principalmente nos exemplos envolvendo o neurônio de McCulloch-Pitts realizados neste trabalho e apresentadas, por exemplo, nas Equações (**??**) e (1.4), para as funções do tipo sigmoides,

permitiu que o algoritmo *backpropagation* pudesse ser aplicado no treinamento de um PMC.

Géron (2019, p.270), comenta que as funções sigmoides foram utilizadas durante muito tempo como analogia ao que se parecia com a curva relativa a ativação de neurônios biológicos mas, atualmente, a mais utilizada é a função ReLU[2], Figura 9.

A Tabela 8 apresenta quatro das funções de ativação amplamente utilizadas na literatura.

Função	**Fórmula**	**Conjunto imagem**
Logística	$\sigma(x) = \dfrac{1}{1+e^{-x}}$	(0, 1)
ReLU	$\text{ReLU}(x) = \begin{cases} x, & \text{se } x > 0, \\ 0, & \text{se } x \leqslant 0 \end{cases}$	$[0, +\infty)$
Tangente Hiperbólica	$\tanh(x) = \dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	(-1, 1)
Degrau	$f(x) = \begin{cases} 1, & \text{se } x \geqslant 0, \\ 0, & \text{se } x < 0 \end{cases}$	{0, 1}

Tabela 8 – Tabelas com algumas funções de ativação. Elaborado pelo autor.

Segundo Widrow e Lehr (1990), as funções de ativação adotadas ditas do tipo sigmoide são aquelas que apresentam uma curva em forma de S quando seu domínio varia de valores próximos a -1 até valores próximos a 1 como pode ser visto na Figura 8.

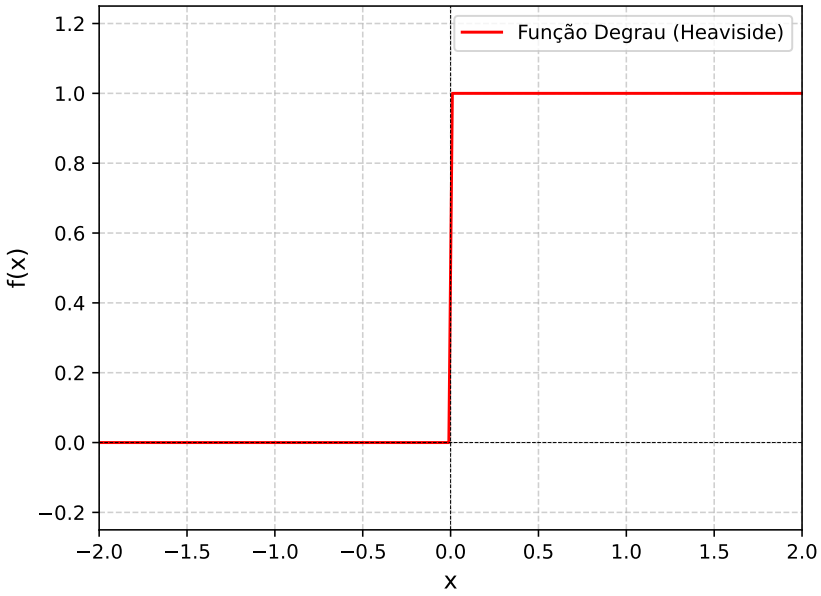


Figura 7 – Gráfico da função degrau, também conhecida como função degrau ou função de Heaviside. Elaborado pelo autor.

[2] Acrônimo para *Rectified Linear Unit*.

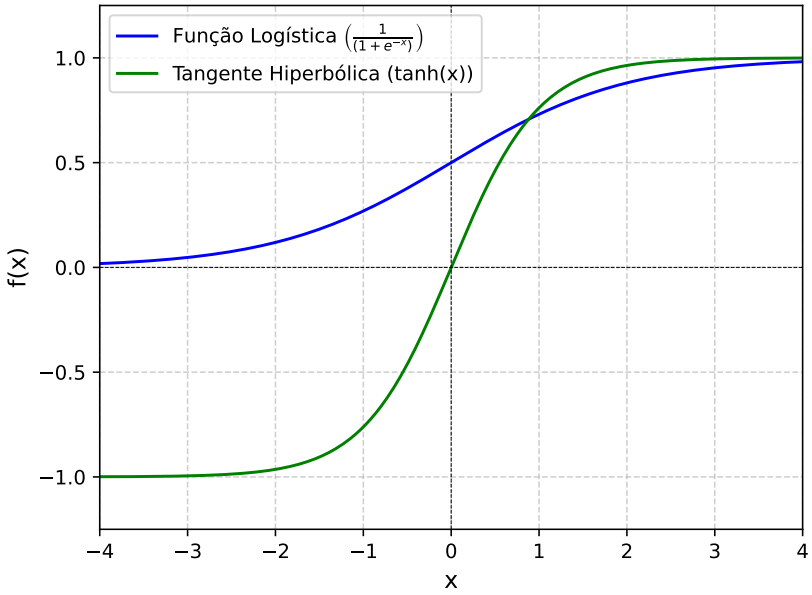


Figura 8 – Exemplos de gráficos de duas funções ditas sigmoidais, de acordo com Widrow e Lehr (1990). Elaborado pelo autor.

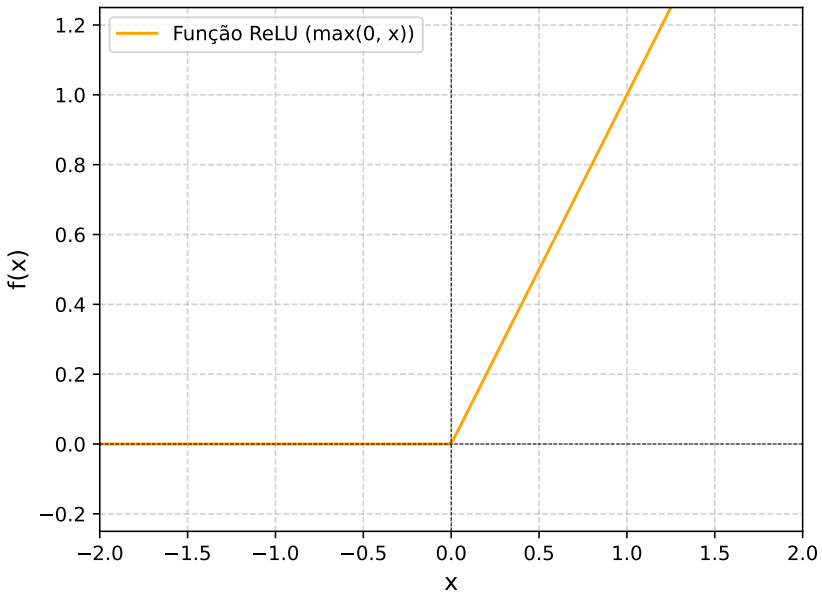


Figura 9 – Gráfico da função ReLU. Elaborado pelo autor.

1.3.1 A fase *Forward*

Devidos aos propósitos didáticos deste texto, a apresentação do algoritmo de treinamento do PMC será desenvolvida com base no modelo simplificado apresentado na Figura 11. A utilização de um exemplo genérico, como o da Figura 10, demandaria um

maior nível de abstração por parte do leitor que poderia ficar preso no entendimento do elevado número de índices que apareceriam neste exemplo generalizado.

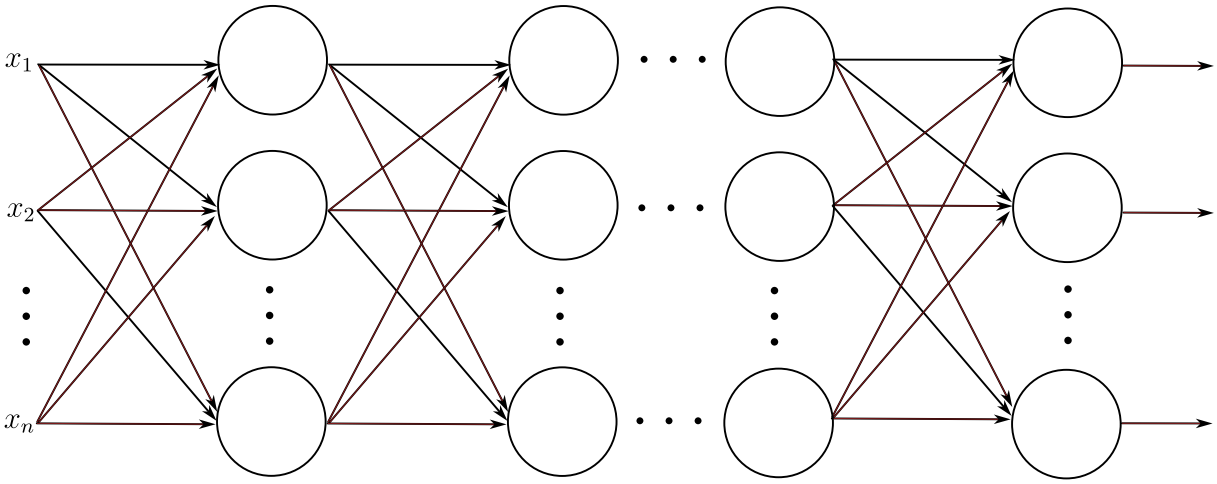


Figura 10 – Perceptron Multicamadas genérico. Elaborado pelo autor.

Caso leitor tenha interesse na versão generalizada desta demonstração, recomenda-se a leitura de Redes Neurais: Princípios e Prática (2001), que é muito bem detalhada, enquanto uma apresentação mais sucinta pode ser vista em 30 years of adaptive neural networks: perceptron, Madaline, and backpropagation (1990).

Será adotado o modo escolhido por Silva, Spatti e Flauzino (2016) ao apresentar um caso particular de PMC para ilustrar o funcionamento do algoritmo de treinamento, através da utilização de um PMC com três camadas, contendo três neurônios na primeira camada (de entrada), dois na segunda (camada oculta) e dois na terceira camada (de saída), como visto na Figura 11.

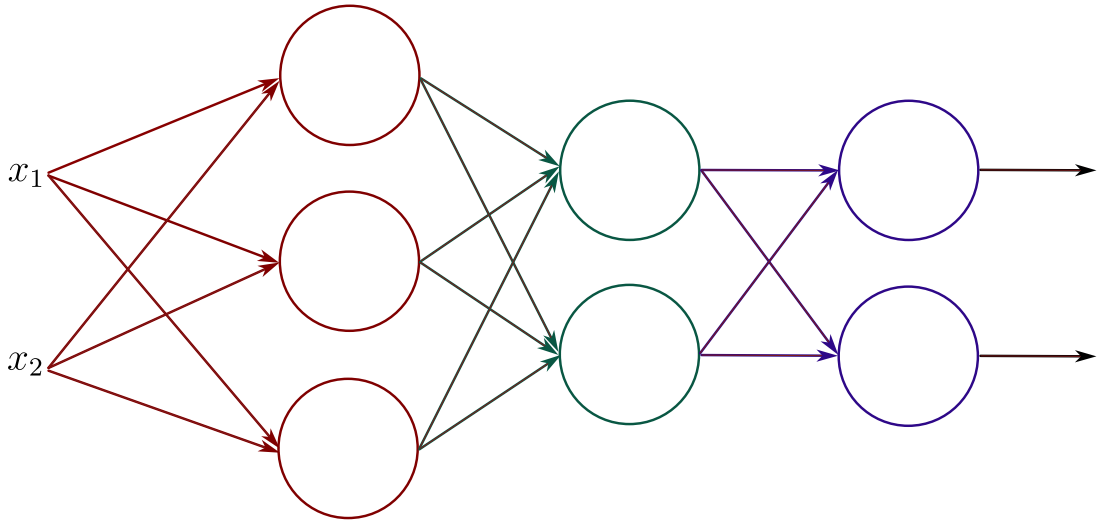


Figura 11 – PMC322 com sentidos da fase *forward* indicados pelas setas. Elaborado pelo autor.

A Figura 11 apresenta não só a rede utilizada mas também o sentido em que os dados são propagados pela rede PMC na fase *forward*.

Antes de prosseguir, deve-se identificar a quais neurônios os pesos sinápticos se referem. As camadas serão ordenadas da esquerda para a direita utilizando a ordem numérica crescente, assim como serão identificados os neurônios de cima para baixo, também em ordem crescente e em ambos começando pelo 1.

Com esta convenção estabelecida, pode-se identificar a qual camada se refere, e a qual neurônio pertencem, os pesos sinápticos. A Figura 12 apresenta os pesos sinápticos em seus respectivos neurônios e camadas.

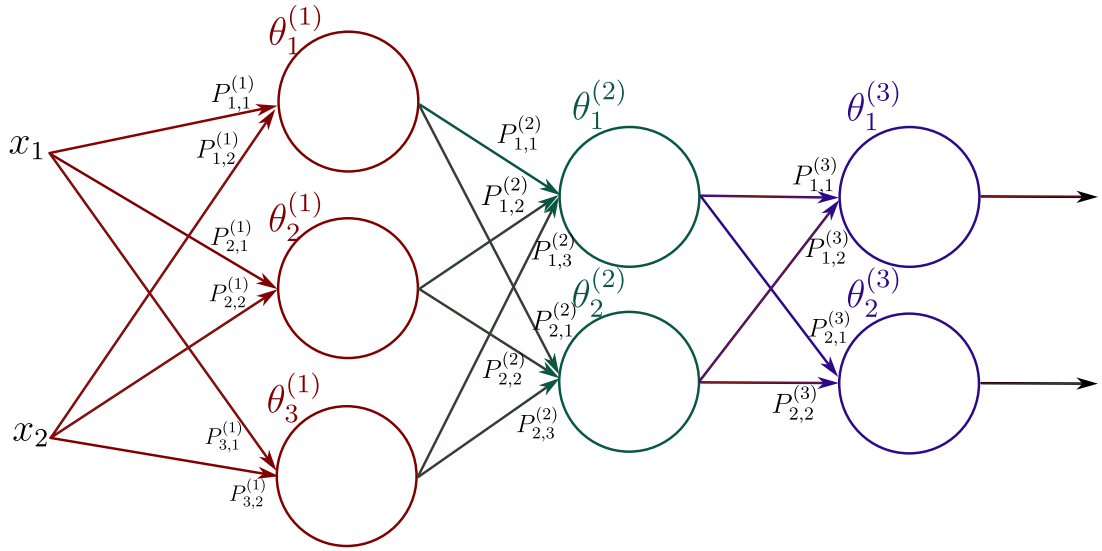


Figura 12 – PMC322 com pesos sinápticos identificados. Elaborado pelo autor.

Por meio da Figura 12 , é possível exemplificar o modo que os pesos estão identificados. O peso genérico $P_{ij}^{(c)}$ é um peso sináptico que pondera a entrada j, advinda da camada $(c-1)$, ao neurônio i da camada (c). Por exemplo, $P_{1,2}^{(2)}$ é um peso sináptico da camada 2, que pondera o sinal proveniente do 2ª neurônio da camada 1 como entrada ao 1º neurônio da camada 2.

Pode-se adotar quaisquer funções mencionadas na tabela 8, com exceção da função degrau. Porém, pretende-se seguir com a notação de função f, sem especificar uma função, com o objetivo de apresentar o modo como os dados são propagados entre as camadas.

Como mencionado anteriormente, será realizado o treinamento em linha. Isto significa que serão assumidos x_1 e x_2 como sendo variáveis de uma amostra específica do conjunto de amostras x. Estas variáveis formarão o conjunto de entradas. Os neurônios da primeira camada as combinam linearmente com seus respectivos pesos sinápticos e limiares para em seguida aplicar a função de ativação. Pode-se escrever esta etapa como

na Equação 1.5.

$$\begin{aligned} Y_1^{(1)} &= f\left(P_{11}^{(1)} \cdot x_1 + P_{12}^{(1)} \cdot x_2 - \theta_1^{(1)}\right) \\ Y_2^{(1)} &= f\left(P_{21}^{(1)} \cdot x_1 + P_{22}^{(1)} \cdot x_2 - \theta_2^{(1)}\right) \\ Y_3^{(1)} &= f\left(P_{31}^{(1)} \cdot x_1 + P_{32}^{(1)} \cdot x_2 - \theta_3^{(1)}\right). \end{aligned} \tag{1.5}$$

Na Equação (1.5), o $Y_j^{(c)}$ representa a saída do neurônio j da camada c, por exemplo, $Y_3^{(1)}$ representa a saída do terceiro neurônio da primeira camada, e, $\theta_j^{(c)}$ representa o limiar do neurônio j da camada c.

A segunda camada receberá as saídas da primeira camada e realizará com elas as combinações lineares com pesos sinápticos e limiares específicos, de modo similar ao que foi realizado para a primeira camada, como é possível ver na Equação (1.6).

$$\begin{aligned} Y_1^{(2)} &= f\left(P_{11}^{(2)} \cdot Y_1^{(1)} + P_{12}^{(2)} \cdot Y_2^{(1)} + P_{13}^{(2)} \cdot Y_3^{(1)} - \theta_1^{(2)}\right) \\ Y_2^{(2)} &= f\left(P_{21}^{(2)} \cdot Y_1^{(1)} + P_{22}^{(2)} \cdot Y_2^{(1)} + P_{23}^{(2)} \cdot Y_3^{(1)} - \theta_2^{(2)}\right). \end{aligned} \tag{1.6}$$

Para a terceira camada, a camada de saída, pode-se fazer como na Equação (1.7).

$$\begin{aligned} Y_1^{(3)} &= f\left(P_{11}^{(3)} \cdot Y_1^{(2)} + P_{12}^{(3)} \cdot Y_2^{(2)} - \theta_1^{(3)}\right) \\ Y_2^{(3)} &= f\left(P_{21}^{(3)} \cdot Y_1^{(2)} + P_{22}^{(3)} \cdot Y_2^{(2)} - \theta_2^{(3)}\right). \end{aligned} \tag{1.7}$$

Uma imagem com todas as saídas com seus respectivos neurônios pode ser vista na Figura 13.

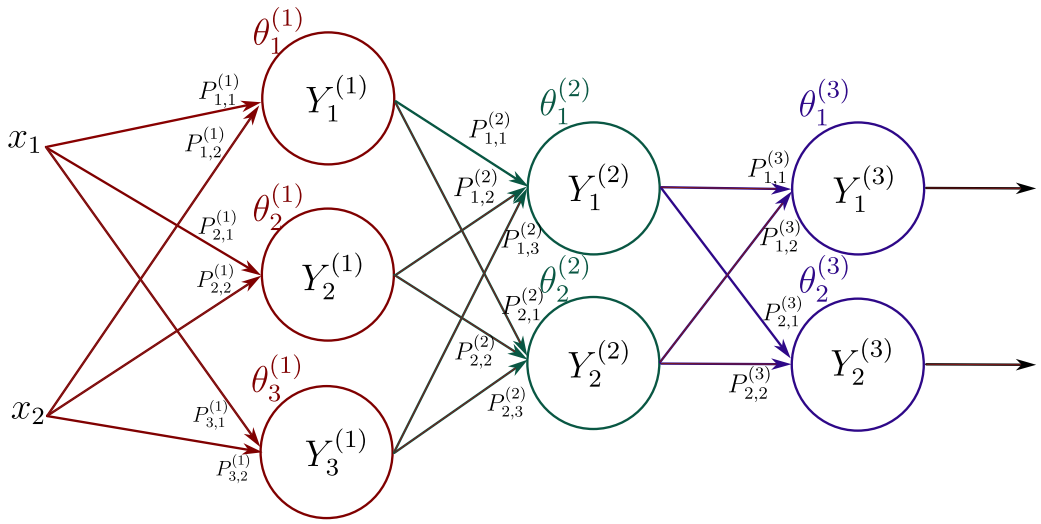


Figura 13 – PMC322 com todas as saídas identificadas com seus respectivos neurônios. Elaborada pelo autor.

As saídas $Y_1^{(3)}$ e $Y_2^{(3)}$ são as que o operador da rede tem acesso. Geralmente, segundo Silva, Spatti e Flauzino (2016), é adotada a prática conhecida como *one of c-classes*, em que, cada neurônio da saída é utilizado para indicar uma das categorias possíveis para os dados. Em outras palavras, esta prática adota para a camada de saída o mesmo número de classes a serem mapeadas com as amostras, caso o problema envolva classificação de dados.

Segundo esta estratégia, para os neurônios da camada de saída a situação específica da função "XOU", por exemplo, a RNA do tipo PMC teria dois neurônios na camada de saída em que cada um deles representaria "1" ou "0", respectivamente. Desta forma, somente um dos neurônios seria ativado como forma de apresentar a classe a qual os dados pertencem.

É possível, ainda, apresentar as saídas dos neurônios da última camada em função dos dados de entrada da primeira camada, como pode ser visto na Equação (1.8). Para facilitar a compreensão dos termos envolvidos nesta Equação, os pesos sinápticos estão coloridos de acordo com a camada a qual pertencem, em vermelhos, verdes e azuis os pesos referentes a primeira, segunda e terceira camadas, respectivamente.

$$
\begin{cases}
Y_1^{(3)} = f\left(
\begin{array}{c}
P_{11}^{(3)} \cdot f\left(
\begin{array}{c}
P_{11}^{(2)} \cdot f\left(P_{11}^{(1)} \cdot x_1 + P_{12}^{(1)} \cdot x_2 - \theta_1^{(1)}\right) + \\
+P_{12}^{(2)} \cdot f\left(P_{21}^{(1)} \cdot x_1 + P_{22}^{(1)} \cdot x_2 - \theta_2^{(1)}\right) + \\
+P_{13}^{(2)} \cdot f\left(P_{31}^{(1)} \cdot x_1 + P_{32}^{(1)} \cdot x_2 - \theta_3^{(1)}\right) - \theta_1^{(2)}
\end{array}
\right) \\
+ \\
P_{12}^{(3)} \cdot f\left(
\begin{array}{c}
P_{21}^{(2)} \cdot f\left(P_{11}^{(1)} \cdot x_1 + P_{12}^{(1)} \cdot x_2 - \theta_1^{(1)}\right) + \\
+P_{22}^{(2)} \cdot f\left(P_{21}^{(1)} \cdot x_1 + P_{22}^{(1)} \cdot x_2 - \theta_2^{(1)}\right) + \\
+P_{23}^{(2)} \cdot f\left(P_{31}^{(1)} \cdot x_1 + P_{32}^{(1)} \cdot x_2 - \theta_3^{(1)}\right) - \theta_2^{(2)}
\end{array}
\right) - \theta_1^{(3)}
\end{array}
\right) \\
Y_2^{(3)} = f\left(
\begin{array}{c}
P_{21}^{(3)} \cdot f\left(
\begin{array}{c}
P_{11}^{(2)} \cdot f\left(P_{11}^{(1)} \cdot x_1 + P_{12}^{(1)} \cdot x_2 - \theta_1^{(1)}\right) + \\
+P_{12}^{(2)} \cdot f\left(P_{21}^{(1)} \cdot x_1 + P_{22}^{(1)} \cdot x_2 - \theta_2^{(1)}\right) + \\
+P_{13}^{(2)} \cdot f\left(P_{31}^{(1)} \cdot x_1 + P_{32}^{(1)} \cdot x_2 - \theta_3^{(1)}\right) - \theta_1^{(2)}
\end{array}
\right) \\
+ \\
P_{22}^{(3)} \cdot f\left(
\begin{array}{c}
P_{21}^{(2)} \cdot f\left(P_{11}^{(1)} \cdot x_1 + P_{12}^{(1)} \cdot x_2 - \theta_1^{(1)}\right) + \\
+P_{22}^{(2)} \cdot f\left(P_{21}^{(1)} \cdot x_1 + P_{22}^{(1)} \cdot x_2 - \theta_2^{(1)}\right) + \\
+P_{23}^{(2)} \cdot f\left(P_{31}^{(1)} \cdot x_1 + P_{32}^{(1)} \cdot x_2 - \theta_3^{(1)}\right) - \theta_2^{(2)}
\end{array}
\right) - \theta_2^{(3)}
\end{array}
\right)
\end{cases}
\tag{1.8}
$$

Para condensar as informações apresentadas pelas combinações lineares, será chamado de $u_i^{(c)}$ a combinação linear referente ao neurônio i na camada c, e fazer a Equação (1.8) se tornar a Equação (1.9).

$$
\begin{cases}
Y_1^{(3)} = f\left(\begin{array}{c}
P_{11}^{(3)} \cdot f\left(\begin{array}{c} P_{11}^{(2)} \cdot f\left(u_1^{(1)}\right) + \\ +P_{12}^{(2)} \cdot f\left(u_2^{(1)}\right) + \\ +P_{13}^{(2)} \cdot f\left(u_3^{(1)}\right) - \theta_1^{(2)} \end{array}\right) \\
+ \\
P_{12}^{(3)} \cdot f\left(\begin{array}{c} P_{21}^{(2)} \cdot f\left(u_1^{(1)}\right) + \\ +P_{22}^{(2)} \cdot f\left(u_2^{(1)}\right) + \\ +P_{23}^{(2)} \cdot f\left(u_3^{(1)}\right) - \theta_2^{(2)} \end{array}\right) - \theta_1^{(3)}
\end{array}\right), \\
Y_2^{(3)} = f\left(\begin{array}{c}
P_{21}^{(3)} \cdot f\left(\begin{array}{c} P_{11}^{(2)} \cdot f\left(u_1^{(1)}\right) + \\ +P_{12}^{(2)} \cdot f\left(u_2^{(1)}\right) + \\ +P_{13}^{(2)} \cdot f\left(u_3^{(1)}\right) - \theta_1^{(2)} \end{array}\right) \\
+ \\
P_{22}^{(3)} \cdot f\left(\begin{array}{c} P_{21}^{(2)} \cdot f\left(u_1^{(1)}\right) + \\ +P_{22}^{(2)} \cdot f\left(u_2^{(1)}\right) + \\ +P_{23}^{(2)} \cdot f\left(u_3^{(1)}\right) - \theta_2^{(2)} \end{array}\right) - \theta_2^{(3)}
\end{array}\right)
\end{cases}
$$

$$
\Leftrightarrow
$$

$$
\begin{cases}
Y_1^{(3)} = f\left(P_{11}^{(3)} \cdot f\left(u_1^{(2)}\right) + P_{12}^{(3)} \cdot f\left(u_2^{(2)}\right) - \theta_1^{(3)}\right), \\
Y_2^{(3)} = f\left(P_{21}^{(3)} \cdot f\left(u_1^{(2)}\right) + P_{22}^{(3)} \cdot f\left(u_2^{(2)}\right) - \theta_2^{(3)}\right)
\end{cases}
$$

$$
\Leftrightarrow
$$

$$
\begin{cases}
Y_1^{(3)} = f\left(u_1^{(3)}\right), \\
Y_2^{(3)} = f\left(u_2^{(3)}\right)
\end{cases}
\tag{1.9}
$$

É importante relembrar que, como f é uma função com conjunto imagem contido em um intervalo real, as saídas são valores reais. A partir do momento em que as saídas apresentadas na Equação (1.9) são obtidas, é iniciada a próxima fase do treinamento, a fase *backward*.

1.3.2 A fase *Backward*

Ao neurônio da rede PMC é atribuída duas funções, sendo primeira delas vista na seção anterior, que não se diferencia muito daquela proposta ao neurônio quando está

atuando isoladamente em uma rede do tipo Adaline ou Perceptron: a combinação linear dos pesos sinápticos com as variáveis e o uso da função de ativação.

A segunda função atribuída ao neurônio é a de realizar os ajustes de seus pesos sinápticos de acordo com a posição que está na arquitetura da rede neural. Estes ajustes são dependentes dos ajustes realizados nos neurônios na camada seguinte ao neurônio em questão.

É nesta fase, chamada *backward*[3] , que com o uso da minimização da função Erro Quadrático, os pesos da camada de saída são ajustados e os fatores utilizados neste ajustes são propagados de trás para frente, ou seja, no sentido dos neurônios da camada de saída para os neurônios da primeira camada, como ilustrado na Figura 14.

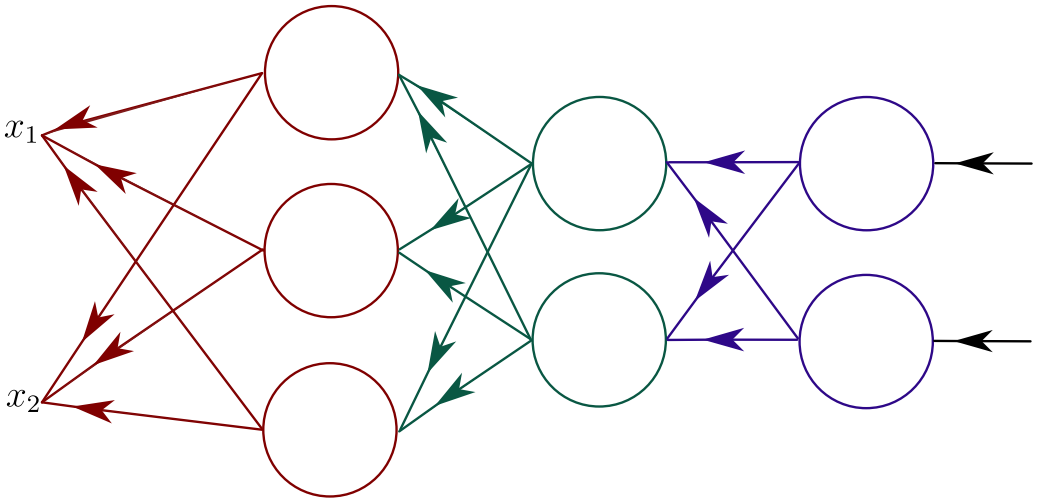


Figura 14 – PMC322 Com sentido indicado do fluxo de informações na etapa *backward* do treinamento. Elaborado pelo autor.

Esta estratégia é adotada pois, no caso do PMC não se tem acesso às saídas de todos os neurônios, mas sim somente às dos neurônios na camada de saída.

Como estas saídas são na verdade funções de outras funções, e assim sucessivamente, até uma função diretamente em função dos primeiros pesos sinápticos, como pode ser visto na Equação (1.8), pode-se basear na fundamentação relativa à composição de funções para pensar no ajuste dos pesos sinápticos.

A função Erro Quadrático que será minimizada é a mesma utilizada no capítulo **??**, porém, será aplicada em mais de um neurônio caso a arquitetura do PMC em questão tenha mais de um neurônio em sua camada de saída.

Como esta função não pode ser aplicada diretamente em função dos pesos sinápticos de camadas anteriores às da saída, o processo de minimização que depende do uso de derivadas, agora também dependerá da regra da cadeia para derivadas. Aqui também

[3] Uma possível tradução para o português seria algo como “retroativo”.

deve-se recorrer ao uso do gradiente, pois, cada camada contém um conjunto de pesos que atuam ao mesmo tempo na função E, Equação (1.10).

Continuando com o exemplo de neurônio PMC322, Figura 14, serão obtidos os fatores a serem utilizados nos ajustes dos pesos sinápticos dos neurônios da camada de saída. Assim como no Adaline, com o objetivo de minimizar o Erro Quadrático, utiliza-se a ideia de otimização associada às derivadas de E em função dos pesos sinápticos da camada, e neurônio, em questão. A Equação (1.10) apresenta o erro quadrático da camada de saída:

$$E^{(3)}(P) = \frac{1}{2}\left[\left(d_1 - Y_1^{(3)}\right)^2 + \left(d_2 - Y_2^{(3)}\right)^2\right]. \tag{1.10}$$

Na Equação (1.10), d_1 e d_2 são as saídas associadas aos valores específicos da amostra que possui x_1 e x_2 como variáveis de entrada.

1.3.2.1 Ajustando os pesos dos neurônios da camada de saída (terceira camada)

O objetivo do processo de treinamento, para cada camada de neurônios, é ajustar os pesos sinápticos de modo a minimizar o erro entre a saída da rede e a saída desejada. Isto implica na minimização da função $E^{(3)}$, dada pela Equação (1.10).

Calculando a derivada de (1.10) em relação aos pesos, P_{ij} da camada de saída tem-se o gradiente do erro quadrático:

$$\nabla E^{(3)} = \frac{\partial E}{\partial P_{i,j}^{(3)}} = \frac{\partial E}{\partial Y_i^{(3)}} \frac{\partial Y_i^{(3)}}{\partial u_i^{(3)}} \frac{\partial u_i^{(3)}}{\partial P_{i,j}^{(3)}}. \tag{1.11}$$

Como está sendo utilizada uma configuração particular de RNA, pode-se atribuir valores aos índices i e j de modo a explorar o os termos da Equação (1.11). Analisando o gradiente em função do peso sináptico $P_{1,1}^{(3)}$, adotando $i = 1$ e $j = 1$, a Equação apresentada em (1.11) resulta na Equação (1.12):

$$\frac{\partial E}{\partial P_{1,1}^{(3)}} = \overbrace{\frac{\partial E}{\partial Y_1^{(3)}}}^{1)} \underbrace{\frac{\partial Y_1^{(3)}}{\partial u_1^{(3)}}}_{2)} \overbrace{\frac{\partial u_1^{(3)}}{\partial P_{1,1}^{(3)}}}^{3)}. \tag{1.12}$$

Analisando termo a termo da regra da cadeia apresentada na Equação (1.12), tem-se que:

$$\begin{aligned} 1) \quad & \frac{\partial E}{\partial Y_1^{(3)}} = -(d_1 - Y_1^{(3)}) \\ 2) \quad & \frac{\partial Y_1^{(3)}}{\partial u_1^{(3)}} = f'(u_1^{(3)}) \\ 3) \quad & \frac{\partial u_1^{(3)}}{\partial P_{1,1}^{(3)}} = Y_1^{(2)}. \end{aligned} \tag{1.13}$$

De (1.13) tem-se que:

$$\frac{\partial E}{\partial P_{1,1}^{(3)}} = -(d_1 - Y_1^{(3)}) \cdot f'(u_1^{(3)}) \cdot Y_1^{(2)}. \tag{1.14}$$

Dessa forma, o fator de correção $\Delta P_{1,1}^{(3)}$, a ser utilizado para ajustar o peso sináptico $P_{1,1}^{(3)}$, deve ser efetuado na direção oposta ao gradiente a fim de minimizar o erro, ou seja:

$$\begin{aligned} \Delta P_{1,1}^{(3)} &= -\eta \cdot \frac{\partial E}{\partial P_{1,1}^{(3)}} \\ &= \eta \cdot (d_1 - Y_1^{(3)}) \cdot f'(u_1^{(3)}) \cdot Y_1^{(2)} \\ &= \eta \cdot \delta_1^{(3)} \cdot Y_1^{(2)}, \end{aligned} \tag{1.15}$$

em que $\delta_1^{(3)} = (d_1 - Y_1^{(3)}) \cdot f'(u_1^{(3)})$ é definido como o gradiente local em relação ao primeiro neurônio da camada de saída.

A constante η define a taxa de aprendizagem para o treinamento e tem proposta semelhante à apresentada para as RNA anteriores ao PMC neste trabalho, ou seja, corresponde ao quanto dos fatores de correção serão incorporados ao novo peso sináptico.

Na segunda linha da Equação (1.13) aparece a derivada da função f, ou seja, a derivada da função de ativação adotada para a rede, cujos valores são apresentados na Tabela 9.

Função	**Derivada**
Logística	$\sigma'(x) = \sigma(x)(1 - \sigma(x))$
ReLU	$\text{ReLU}'(x) = \begin{cases} 1, & \text{se } x > 0, \\ 0, & \text{se } x \leqslant 0 \end{cases}$
Tangente Hiperbólica	$\tanh'(x) = 1 - \tanh^2(x)$
Degrau	$f'(x) = \begin{cases} \text{indefinido}, & \text{em } x = 0, \\ 0, & \forall x \in \mathbb{R} - \{0\} \end{cases}$

Tabela 9 – Tabela com as derivadas de algumas funções de ativação. Elaborado pelo autor.

De modo similar ao que foi realizado com o peso sináptico $P_{1,1}^{(3)}$, são obtidos os fatores de correção para todos os pesos sinápticos do primeiro neurônio da terceira camada

(camada de saída), como pode ser visto na Equação (1.16):

$$\begin{aligned} \Delta P_{1,1}^{(3)} &= \eta \cdot \delta_1^{(3)} \cdot Y_1^{(2)} \\ \Delta P_{1,2}^{(3)} &= \eta \cdot \delta_1^{(3)} \cdot Y_2^{(2)}. \end{aligned} \tag{1.16}$$

Desta forma, a atualização que os pesos sinápticos do primeiro neurônio da terceira camada será atualizado pela Equação (1.17), com o índice j representando a origem da informação de atualização.

$$P_{1,j}^{(3)(novo)} = P_{1,j}^{(3)(antigo)} + \eta \cdot \delta_1^{(3)} \cdot Y_j^{(2)}. \tag{1.17}$$

Analogamente ao que foi exposto nas Equações (1.16) e (1.17), com as devidas atualizações de índices, os pesos sinápticos para o segundo neurônio da terceira camada é atualizado.

De modo geral, é possível escrever para os neurônios da camada de saída a regra de ajustes apresentada na Equação (1.18):

$$P_{i,j}^{(3)(novo)} = P_{i,j}^{(3)(antigo)} + \eta \cdot \delta_i^{(3)} \cdot Y_j^{(2)}, \tag{1.18}$$

em que os índices i e j dizem respeito ao destino e a sua origem dos pesos em atualização, respectivamente.

Finalizados os ajustes dos pesos sinápticos de todos os neurônios da camada de saída, ou seja, da terceira camada, deve-se iniciar os ajustes dos neurônios da camada anterior a esta, ou seja, a segunda camada.

1.3.2.2 Ajustando os pesos dos neurônios da segunda camada

Como não se tem acesso às saídas destes neurônios para que se possa efetuar diretamente os ajustes, será considerado o fato das saídas da RNA do tipo PMC serem, basicamente, funções compostas por outras funções, e assim utilizar a regra da cadeia para ajustar os neurônios de camadas ocultas.

Deste modo, calculando-se gradiente da função erro quadrático E, agora em função dos pesos sinápticos da segunda camada, obtém-se a Equação (1.19):

$$\nabla E^{(2)} = \frac{\partial E}{\partial P_{i,j}^{(2)}} = \frac{\partial E}{\partial Y_i^{(2)}} \frac{\partial Y_i^{(2)}}{\partial u_i^{(2)}} \frac{\partial u_i^{(2)}}{\partial P_{i,j}^{(2)}}. \tag{1.19}$$

Utilizando a mesma estratégia utilizada anteriormente, serão adotados os valores para os índices com $i = 1$ e $j = 1$, ou seja, serão detalhados os processos para atualização do peso sináptico $P_{1,1}^{(2)}$ e a saída $Y_1^{(2)}$.

Com isto, obtém-se:

$$\frac{\partial E}{\partial P_{1,1}^{(2)}} = \underbrace{\frac{\partial E}{\partial Y_1^{(2)}}}_{1)} \overbrace{\frac{\partial Y_1^{(2)}}{\partial u_1^{(2)}}}^{2)} \underbrace{\frac{\partial u_1^{(2)}}{\partial P_{1,1}^{(2)}}}_{3)}. \tag{1.20}$$

Os termos 2) e 3) da Equação (1.20) são análogos aos da terceira camada, com os devidos ajustes de índices e, são dados pela Equação (1.21):

$$\begin{aligned} 2)\, &\frac{\partial Y_1^{(2)}}{\partial u_1^{(2)}} = f'(u_1^{(2)}) \\ 3)\, &\frac{\partial u_1^{(2)}}{\partial P_{1,1}^{(2)}} = Y_1^{(1)}. \end{aligned} \tag{1.21}$$

Como a função E não recebe diretamente a saída $Y_1^{(2)}$, a derivada parcial apresentada em 1) precisará de algumas manipulações. Primeiramente, deve-se lembrar que $u_1^{(3)}$ e $u_2^{(3)}$ recebem $Y_1^{(2)}$ e que com ele operam com seus respectivos pesos sinápticos $P_{1,1}^{(3)}$ e $P_{2,1}^{(3)}$, como pode ser visto na Equação (1.7). Desta forma, a derivada parcial 1) pode ser reescrita através da regra da cadeia dada pela Equação (1.22):

$$\begin{aligned} 1)\quad \frac{\partial E}{\partial Y_1^{(2)}} &= \frac{\partial E}{\partial u_1^{(3)}} \frac{\partial u_1^{(3)}}{\partial Y_1^{(2)}} + \frac{\partial E}{\partial u_2^{(3)}} \frac{\partial u_2^{(3)}}{\partial Y_1^{(2)}} \\ &= \frac{\partial E}{\partial Y_1^{(3)}} \frac{\partial Y_1^{(3)}}{\partial u_1^{(3)}} P_{1,1}^{(3)} + \frac{\partial E}{\partial Y_1^{(3)}} \frac{\partial Y_1^{(3)}}{\partial u_2^{(3)}} P_{2,1}^{(3)}. \end{aligned} \tag{1.22}$$

Os resultados das regras das cadeias apresentadas em ambos os fatores, em preto e em azul, na Equação (1.22) podem ser obtidos por meio da aplicação do gradiente realizado na Equação (1.11). Isto significa que estes fatores são, na verdade, os mesmos que apareceram na Equação (1.13) como é possível ver na Equação (1.23):

$$\begin{aligned} \frac{\partial E}{\partial Y_1^{(3)}} \frac{\partial Y_1^{(3)}}{\partial u_1^{(3)}} &= -(d_1 - Y_1^{(3)}) \cdot f'(u_1^{(3)}) = \delta_1^{(3)} \\ \frac{\partial E}{\partial Y_1^{(3)}} \frac{\partial Y_1^{(3)}}{\partial u_2^{(3)}} &= -(d_2 - Y_2^{(3)}) \cdot f'(u_2^{(3)}) = \delta_2^{(3)}. \end{aligned} \tag{1.23}$$

Desta forma, é possível reescrever a Equação (1.22) como a Equação (1.24).

$$1)\quad \frac{\partial E}{\partial Y_1^{(2)}} = \delta_1^{(3)} P_{1,1}^{(3)} + \delta_2^{(3)} P_{2,1}^{(3)}. \tag{1.24}$$

Um detalhe importante é que os pesos sinápticos dos neurônios da terceira camada que recebem a saída do primeiro neurônio da segunda camada, pesos $P_{1,1}^{(3)}$ e $P_{2,1}^{(3)}$ respectivamente, já estão atualizados e influenciarão diretamente na atualização dos pesos sinápticos dos neurônios da segunda camada aos quais estão conectados.

Unindo os fatores 1), 2) e 3), pode-se reescrever o gradiente aplicado na função E para o peso sináptico $P_{1,1}^{(2)}$ da segunda camada como:

$$\frac{\partial E}{\partial P_{1,1}^{(2)}} = \left(\delta_1^{(3)} P_{1,1}^{(3)} + \delta_2^{(3)} P_{2,1}^{(3)}\right) \cdot f'(u_1^{(2)}) \cdot Y_1^{(1)}. \tag{1.25}$$

A manipulação foi longa, mas, realizando passos similares e alterando somente os índices envolvidos, é possível escrever as variações no erro quadrático em função dos pesos sinápticos para o primeiro neurônio da segunda camada através das equações (1.26):

$$\begin{aligned}
\frac{\partial E}{\partial P_{1,1}^{(2)}} &= \left(\delta_1^{(3)} P_{1,1}^{(3)} + \delta_2^{(3)} P_{2,1}^{(3)}\right) \cdot f'(u_1^{(2)}) \cdot Y_1^{(1)} \\
\frac{\partial E}{\partial P_{2,1}^{(2)}} &= \left(\delta_1^{(3)} P_{1,1}^{(3)} + \delta_2^{(3)} P_{2,1}^{(3)}\right) \cdot f'(u_1^{(2)}) \cdot Y_2^{(1)} \\
\frac{\partial E}{\partial P_{3,1}^{(2)}} &= \left(\delta_1^{(3)} P_{1,1}^{(3)} + \delta_2^{(3)} P_{2,1}^{(3)}\right) \cdot f'(u_1^{(2)}) \cdot Y_3^{(1)}.
\end{aligned} \tag{1.26}$$

Como o fator $\left(\delta_1^{(3)} P_{1,1}^{(3)} + \delta_2^{(3)} P_{2,1}^{(3)}\right) \cdot f'(u_1^{(2)})$ é comum em todas os ajustes que serão realizados, pode-se chamá-lo de $\delta_1^{(2)}$ e reescrever as equações na Equação (1.26) em função do peso a ser atualizado e renomeá-la como o fator de correção destes pesos, ponderado pela taxa de aprendizado η, como na Equação (1.27):

$$\Delta P_{1,j}^{(2)} = \eta \cdot \delta_1^{(2)} \cdot Y_j^{(1)}. \tag{1.27}$$

Seguindo os mesmos passos para o segundo neurônio da segunda camada, é possível obter o fator de ajustes de seus pesos sinápticos como na Equação (1.28):

$$\Delta P_{2,j}^{(2)} = \eta \cdot \delta_2^{(2)} \cdot Y_j^{(1)}, \tag{1.28}$$

em que $\delta_2^{(2)} = \left(\delta_1^{(3)} P_{1,2}^{(3)} + \delta_2^{(3)} P_{2,2}^{(3)}\right) \cdot f'(u_2^{(2)})$.

E assim, os pesos sinápticos dos neurônios da segunda camada serão ajustados de acordo com a Equação (1.29)

$$P_{i,j}^{(2)(novo)} = P_{i,j}^{(2)(antigo)} + \eta \cdot \delta_i^{(2)} \cdot Y_j^{(2)}. \tag{1.29}$$

1.3.2.3 Ajustando os pesos dos neurônios da primeira camada

De modo similar ao que foi feito com o primeiro neurônio da segunda camada, para o primeiro neurônio da primeira camada também serão detalhadas todas as etapas de aplicação de gradiente em função dos pesos sinápticos $P_{1,0}^{(1)}$,$P_{1,1}^{(1)}$,$P_{1,2}^{(1)}$ e $P_{1,3}^{(1)}$ ligados a este neurônio e a utilização de uma função adequada para a regra da cadeia em função destes pesos sinápticos.

Em resumo, o fator de ajustes $\delta_1^{(1)}$ para o peso sináptico $P_{1,1}^{(1)}$ do primeiro neurônio da primeira camada pode ser expresso como na Equação (1.30).

$$\delta_1^{(1)} = \left(\delta_1^{(2)} \cdot f'(u_1^{(2)}) \cdot P_{1,1}^{(2)} + \delta_2^{(2)} \cdot f'(u_2^{(2)}) \cdot P_{2,1}^{(2)}\right) f'(u_1^{(1)}) \cdot x_1, \tag{1.30}$$

que foi obtido por meio de

$$\nabla E^{(1)} = \frac{\partial E}{\partial P_{1,1}^{(1)}} = \frac{\partial E}{\partial Y_1^{(1)}} \frac{\partial Y_1^{(1)}}{\partial u_1^{(1)}} \frac{\partial u_1^{(1)}}{\partial P_{1,1}^{(1)}}. \tag{1.31}$$

Assim, estendendo para todos os pesos sinápticos do primeiro neurônio na primeira camada o que foi realizado, pode-se atualizá-los de acordo com as equações de (1.32):

$$\begin{aligned} P_{1,1}^{(1)(novo)} &= P_{1,1}^{(1)(antigo)} + \eta \cdot \delta_1^{(1)} \cdot x_1 \\ P_{1,2}^{(1)(novo)} &= P_{1,2}^{(1)(antigo)} + \eta \cdot \delta_1^{(1)} \cdot x_2. \end{aligned} \tag{1.32}$$

Generalizando para todos os pesos sinápticos de todos os neurônios da primeira camada, é possível escrever sua regra de aprendizado como na Equação (1.33):

$$P_{i,j}^{(1)(novo)} = P_{i,j}^{(1)(antigo)} + \eta \cdot \delta_i^{(1)} \cdot x_j. \tag{1.33}$$

A Equação (1.33) contém x_j, em que j varia de 1 a 2, representando as variáveis de entradas de uma amostra específica.

Construiu-se a Tabela 10 apresentando os fatores utilizados na correção dos pesos sinápticos $P_{1,1}^{(c)}$ dos primeiros neurônios em cada camada c.

Estão identificados, em azul, os fatores de correção da terceira camada, em verde, os fatores de correção na segunda camada, e em vermelho, os fatores de correção da primeira camada juntamente com suas respectivas saídas dos neurônios, ou seja, cada cor indica a camada a qual aquelas informações pertencem.

$P_{1,1}^{(c)}$		
Camada (c)	Fator de correção	Dependência
3	$\delta_1^{(3)}$	$(d_1 - Y_1^{(3)}) \cdot f'(u_1^{(3)}) \cdot Y_1^{(2)}$
2	$\delta_1^{(2)}$	$\left(\delta_1^{(3)} \cdot P_{1,1}^{(3)} + \delta_2^{(3)} \cdot P_{2,1}^{(3)}\right) \cdot f'(u_1^{(2)}) \cdot Y_1^{(1)}$
1	$\delta_1^{(1)}$	$\left(\delta_1^{(2)} \cdot f'(u_1^{(2)}) \cdot P_{1,1}^{(2)} + \delta_2^{(2)} \cdot f'(u_2^{(2)}) \cdot P_{2,1}^{(2)}\right) f'(u_1^{(1)}) \cdot x_1$

Tabela 10 – Tabela apresentando os fatores de correção utilizados nos primeiros neurônios de cada camada. Elaborada pelo autor.

A notação matemática para todo este processo pode, em um primeiro momento, parecer excessiva e confusa. Porém, o que deve ser absorvido disto tudo é a forma como os ajustes são realizados nos pesos sinápticos e o modo que estes ajustes dependem daqueles

que já foram realizados nas camadas seguintes. A Figura 15 apresenta as relações existentes entre os fatores de correção $\delta_i^{(c)}$.

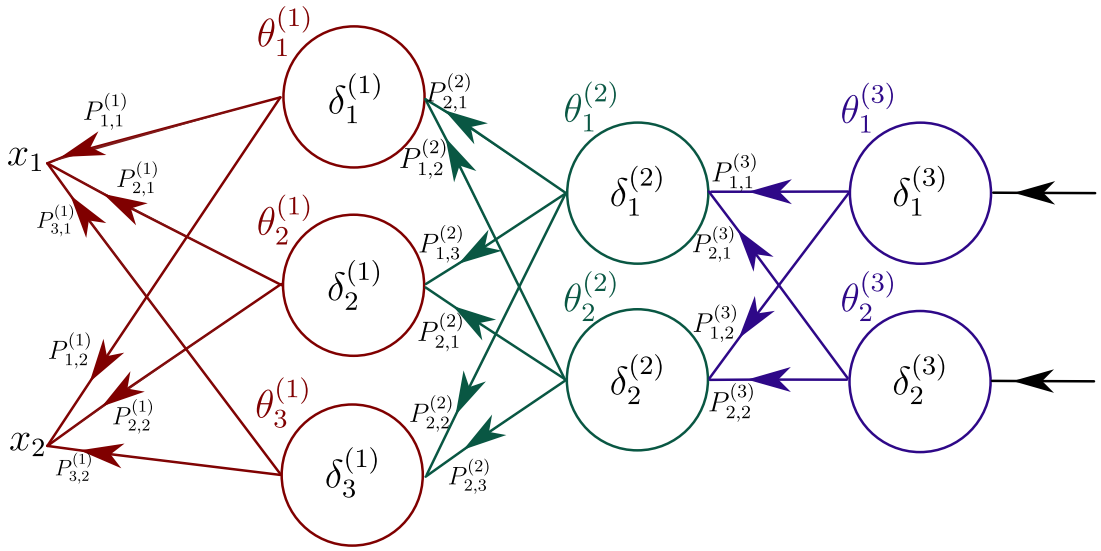


Figura 15 – Disposição dos fatores de correção obtidos na fase *backward* em seus respectivos neurônios. Elaborado pelo autor.

Por meio da Figura 15 é possível observar que, para ajustar o peso sináptico dos neurônios da primeira camada, deve-se primeiro passar pelos ajustes realizados com os pesos dos neurônios que receberão as ponderações realizadas por ele na fase *forward*.

Realizados os ajustes nos pesos sinápticos de todos os neurônios da primeira camada, é encerrada a fase *backward* e o *backpropagation* conclui uma época de treinamento. A partir de então, mantendo o treinamento em linha, outras variáveis do conjunto de entradas são fornecidas à rede e o processo todo, a fase *forward* seguida da *backward*, se repete.

O *backpropagation* não possui um critério universal de parada. Isto significa que o treinamento pode ser parado ao estabelecer-se um número finito de épocas, na observância da acurácia da rede, ou seja, de quantas saídas são apresentadas corretamente ao se comparar com as saídas desejadas, ou algum outro método escolhido.

De fato, Silva, Spatti e Flauzino (2016), Haykin (2001) e Géron (2019) apresentam várias modificações e aperfeiçoamentos para casos específicos de treinamentos envolvendo o *backpropagation*. De um modo geral, é esperado uma convergência da rede para valores ótimos dos pesos sinápticos, ou seja, que os pesos obtidos permitam que a rede efetue com excelência o papel ao qual foi aplicada.

Ainda é possível citar Widrow e Lehr (1990) em que

> "Os algoritmos iterativos descritos neste documento foram todos projetados de acordo com um único princípio subjacente. Essas técnicas (...) baseiam-se no princípio da perturbação mínima: Adaptar-se para reduzir o erro de saída do padrão de treinamento atual, com o mínimo de perturbação das respostas já aprendidas." (Widrow; Lehr, 1990, p.1423, tradução nossa).[4]

1.4 Implementando o Perceptron Multicamadas em Python

A implementação de uma rede PMC em *Python* pode ser um tanto desafiadora. As linhas de código necessárias para uma implementação de modo generalizado, permitindo ajustes na quantidade de neurônios e camadas com o uso de parâmetros e com poucas modificações no código tomaria uma boa parte desta seção. Devido a isto opta-se por utilizar uma biblioteca que contenha uma classe, ou função, que execute o treinamento.

A biblioteca `scikit-learn` (Pedregosa et al., 2011) contém a classe `MLPClassifier`. Esta classe é utilizada em situações envolvendo a classificação dos dados em categorias. Esta biblioteca também fornece a classe `MLPRegressor`, voltado para uso em problemas envolvendo a interpolação dos dados das amostras por uma superfície, pois as saídas desta classe produzem valores contínuos (ver Figura 16a).

A classe `MLPClassifier` produz saídas discretas e portanto é, dentre as duas, a mais adequada para o contexto de categorização. Esta classe não utiliza a função erro quadrático E em seu algoritmo, mas sim a função Regressão Logística[5] L . Primeiramente as saídas da função L são utilizadas como probabilidades $y_{i,j}$ que se relacionarão com as respectivas saídas desejadas $d_{i,j}$ de acordo com a Equação (1.34):

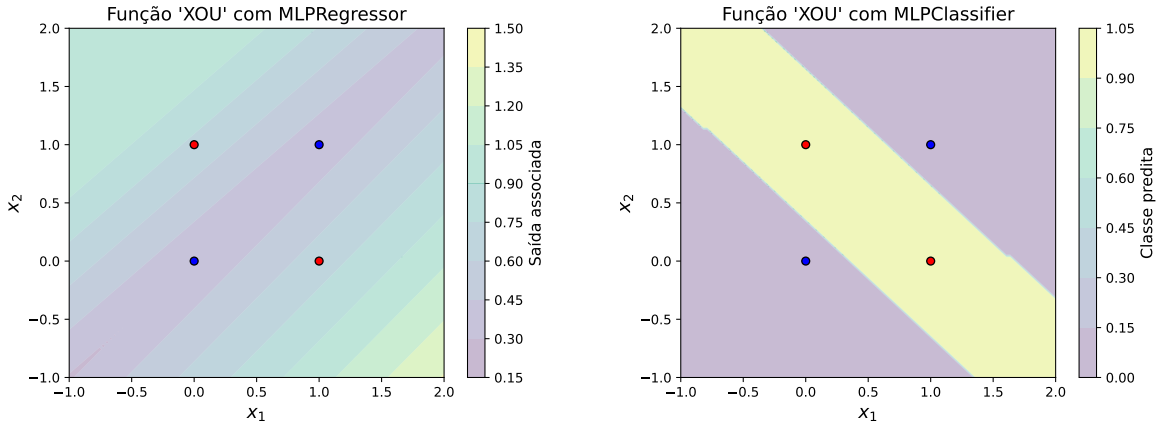
$$L(P) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}\left[d_{ij}\log(y_{i,j}) + (1-d_{ij})\log(1-y_{i,j})\right]. \tag{1.34}$$

Na Equação (1.34), m é o número de amostras, n é o número de variáveis por amostras, i representa a amostra em questão e j sua respectiva saída. Mais sobre a função Regressão logística pode ser visto em Mãos à Obra Aprendizado de Máquina com Scikit-Learn & TensorFlow (2019).

Na prática, aderindo a esta substituição da função E por L, sua otimização por meio do algoritmo de treinamento *backpropagation* permitirá obter ao invés de uma superfície, fronteiras de decisão, ao se utilizar o `MLPClassifier` (ver Figura 16b).

[4] No original, sem edições, "The iterative algorithms described in this paper are all designed in accord with a single underlying principle. These techniques-the two LMS algorithms, Mays's rules, and the Perceptron procedure for training a single Adaline, the MRI rule for training the simple Madaline, as well as MRII, MRIII, and backpropagation techniques for training multilayer Madalines-all rely upon the principle of minimal disturbance: Adapt to reduce the output error for the current training pattern, with minimal disturbance to responses already learned."

[5] Também chamada de *log loss*.



(a) Função booleana "XOU" implementada com `MLPRegressor`.

(b) Função booleana "XOU" implementada com `MLPClassifier`.

Figura 16 – Função booleana "XOU" implementada com as classes `MLPRegressor` e `MLPClassifier`. Elaborados pelo autor.

1.4.1 Implementação da função booleana "XOU"

Com o uso desta biblioteca e estas duas classes implementa-se, por exemplo, a função booleana "XOU". A implementação que é realizada neste trabalho foi utilizada para gerar a Figura 16.

Primeiramente, realiza-se a importação as bibliotecas, como no Algoritmo 1:

```
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import accuracy_score
```

Algoritmo 1 – Importando as bibliotecas para o treinamento do PMC em Python. Elaborada pelo autor.

Em seguida, definem-se as matrizes: x das entradas e d das saídas desejadas, como no Algoritmo 2:

```
# Definindo a tabela verdade XOU em 2 dimensões
x = np.array([[0,0],[0,1],[1,0],[1,1]])
# Saída esperada: 1 quando há um número ímpar de 1 na entrada.
d = np.array([0,1,1,0])
```

Algoritmo 2 – Fornecendo as matrizes de entradas x e de saídas desejadas d para o treinamento do PMC. Elaborada pelo autor.

Utilizando inicialmente a classe `MLPRegressor`, que permite a customização de

diferentes parâmetros internos[6]. Para este estudo de caso serão configurados: a função de ativação; o estado inicial dos pesos sinápticos; como será a otimização realizada pelo *backpropagation*; e o número máximo de iterações da rede. A sequência de comandos apresentada no Algoritmo 3 resumem os parâmetros de inicialização da rede.

```
pmc = MLPRegressor(
    #uma camada escondida com 2 neurônios
    hidden_layer_sizes=(2),
    #função de ativação
    activation='relu',
    #Modo que os gradientes das funções serão utilizados
    solver='sgd',
    #Define o bloco de treinamento do tamanho das amostras
    batch_size=len(x),
    #Semente aleatória para os pesos sinápticos
    random_state=28,
    #Define o número máximo de iterações
    max_iter=20000,
    #Fornece o progresso do treinamento.
    verbose= True
                )
```

Algoritmo 3 – Configurações dos parâmetros do `MLPRegressor`. Elaborado pelo autor.

Alguns métodos e parâmetros de ajustes da rede neural são mais eficientes em determinados contextos, como pode ser visto na documentação da classe `MLPCLassifier`, e também levam em consideração a estratégia de treinamento adotada, se é em lotes, mini-lotes ou em linha, de acordo com os dados fornecidos. Aqui será adotado o treinamento em blocos, ou lotes, e para isto será utilizado o parâmetro `batch_size=len(x)`, que determina o tamanho dos blocos igual ao número de amostras fornecido por `len(x)`.

Fornecer a semente aleatória `random_state` permite que os resultados sejam reproduzíveis com os mesmos parâmetros. Mesmo em 1990, no artigo 30 years of adaptive neural networks: perceptron, Madaline, and backpropagation (1990), os autores já mencionam o quanto os parâmetros iniciais interferem no treinamento da rede como um todo. Há indicações de que determinadas características de conjuntos de pesos sinápticos iniciais favorecem ou não a convergência da RNA em questão e, adotar um parâmetro fixo para `random_state` nos permite reproduzir os resultados obtidos com os pesos iniciais em condições semelhantes todas as vezes que o código for executado.

[6] Uma descrição completa desta classe pode ser vista em https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html. Acessado em 25 de fevereiro de 2025.

De fato, o ajuste dos parâmetros, bem como o número de camadas e neurônios, ainda são objetos de estudo e a escolha destes se baseia na eficiência que apresentam para um mesmo tipo de problema em condições específicas.

O Algoritmo 4 resume a sequência de comandos para treinamento, teste e verificação da acurácia da rede neural após a implementação da rede, cujo resultado é apresentado no Algoritmo 5.

```
# Treinando a rede
pmc.fit(x, d)
# Testando a rede com as amostras
t = pmc.predict(x)
#Transformando as saídas contínuas em discretas
y=t.round()
#Apresenta as saídas obtidas com os pesos treinados
print("Saídas calculadas:", y)
# Avaliando os acertos da rede
acuracia = accuracy_score(d, y)
#Apresenta a porcentagem de acertos
print(f"Acurácia:{acuracia}*100")
```

Algoritmo 4 – Linhas de código executando o treinamento do `MLPRegressor`. Elaborado pelo autor.

```
Iteration 499, loss = 0.05097980
Training loss did not improve more than tol=0.000100 for 10
consecutive epochs. Stopping.
Saídas calculadas: [0. 1. 1. 0.]
Acurácia :100.0%
```

Algoritmo 5 – Saída apresentada ao executar o código fornecido para o treinamento com o `MLPRegressor`. Elaborada pelo autor.

Os resultados após o treinamento podem ser visualizados na Figura 16a. Esta figura apresenta uma série de curvas de nível, separando o plano que contém as entradas em diferentes regiões. Isto ocorreu devido ao fato da saída do `MLPRegressor` ser contínua.

As linhas de código utilizadas para gerar as imagens 16 não serão apresentadas no texto, porém estão disponíveis no caderno do Jupyter Notebook, fornecido em conjunto a este trabalho.

Para implementar a função "XOU", na classe `MLPClassifier`, é possível simplesmente trocar o nome da classe no algoritmo apresentado no Algoritmo 3 e escrever o código para o treinamento do PMC como no Algoritmo 6. A escolha da função de ati-

vação também depende da finalidade a qual a RNA será aplicada, e isto demanda uma inspeção detalhada por parte do executor (Géron, 2019). Neste caso, foi adotada a função tangente hiperbólica pois foi a que produziu melhores resultados neste treinamento, além da mudança na semente aleatória.

```
pmc = MLPClassifier(
    #uma camada escondida com 2 neurônios
    hidden_layer_sizes=(2),
    #função de ativação
    activation='tanh',
    #Modo que os gradientes das funções serão utilizados
    solver='sgd',
    #Define o bloco de treinamento do tamanho das amostras
    batch_size=len(x),
    #Semente aleatória para os pesos sinápticos
    random_state=4,
    #Define o número máximo de iterações
    max_iter=20000,
    #Fornece o progresso do treinamento.
    verbose= True
                    )
```

Algoritmo 6 – Configurações dos parâmetros do `MLPClassifier`. Elaborado pelo autor.

É possível então executar o mesmo algoritmo conforme descrito no Algoritmo 3 com uma pequena alteração no que diz respeito ao arredondamento das saídas produzidas pela rede, que não será mais necessário, e apresentar a sequência de instruções para o treinamento como no Algoritmo 7.

```
# Treinando a rede
pmc.fit(x, d)
# Testando a rede com as amostras
y = pmc.predict(x)
#Apresenta as saídas obtidas com os pesos treinados
print("Saídas calculadas:", y)
# Avaliando os acertos da rede
acuracia = accuracy_score(d, y)
#Apresenta a porcentagem de acertos
print("Acurácia:", acuracia)
```

Algoritmo 7 – Linhas de código executando o treinamento do `MLPClassifier`. Elaborado pelo autor.

O treinamento do PMC com esta classe permite gerar a imagem apresentada na Figura 16b, na qual as fronteiras são representadas por linhas delimitando as regiões de mesma cor que apresentam a mesma saída e, ao executar os códigos do Algoritmo 7, a saída apresentada será como no Algoritmo 8.

```
Iteration 3147, loss = 0.25937596
Training loss did not improve more than tol=0.000100 for 10
consecutive epochs. Stopping.
Saídas calculadas: [0 1 1 0]
Acurácia :100.0%
```

Algoritmo 8 – Saída apresentada pelo PMC que implementa a função booleana "XOU" em duas dimensões. Elaborado pelo autor.

1.4.2 Utilizando o PMC em três categorias: As flores Íris

A possibilidade de um PMC conter em sua camada de saída mais de um neurônio, implica em uma amostra poder ser classificada para uma dentre três, ou mais, categorias. Esta situação pode ser exemplificada com os dados das flores Íris, contidos em Fisher (1988).

Como foi discutido na seção **??**, existem três categorias neste conjunto de dados, sendo duas destas não-linearmente separáveis, e é possível realizar o treinamento do PMC com os dados relativos somente às pétalas ou às sépalas das flores, de modo que se possa apresentar a visualização das fronteiras de decisão que esta RNA estabelecerá a partir dos pesos sinápticos treinados em cada caso.

Para utilizar as flores pode-se adotar a arquitetura para o PMC como no Algoritmo 9 que contém duas camadas ocultas com quatro neurônios cada. Para a camada de saída, a classe `MLPClassifier` atribuirá automaticamente três neurônios, um para cada categoria das flores.

```
#Duas camadas com 4 neurônios cada
pmc = MLPClassifier(hidden_layer_sizes=(4,4),
    #Máximo de iterações
    max_iter=10000,
    #função de ativação tangente hiperbólica
    activation='tanh',
    #Define o uso do Gradiente descendente estocástico
    solver='sgd',
    #Define o bloco de treinamento do tamanho da matriz de amostras
    batch_size=len(x),
    #Define a tolerância da diferença dos erros em épocas sucessivas
    tol=0.00001,
    #Define a semente aleatória para inicialização dos pesos sinápticos
    random_state=5
                    )
```

Algoritmo 9 – Configuração do `MLPClassifier` para treinar o PMC com os dados das flores Íris. Elaborado pelo autor.

A matriz de amostras x contendo os respectivos dados das medidas das flores, em centímetros, pode ser construída a partir dos dados fornecidos em Fisher (1988) ou por meio do comando `sklearn.datasets.load_iris()`[7]. Estes dados também são apresentados, explicitamente, no respectivo caderno interativo Jupyter Notebook que tem o link fornecido no capítulo **??**, que também pode ser acessado pelo site Oliveira (2025).

A matriz de saídas desejadas d pode ser construída de forma similar à matriz de amostras x e, para a realização deste treinamento, a identificação de cada Íris Setosa, Íris Versicolor e Íris Virgínica foi alterada, respectivamente, para 0, 1 e 2. Nos dados disponibilizados, as primeiras cinquenta amostras são de Íris Setosa, seguidas por outras cinquenta de Íris Versicolor, e por último as cinquenta amostras de Íris Virgínica

Após o fornecimento da matriz x de amostras, utiliza-se o Algoritmo 10 para treinar o PMC.

[7] Como pode ser visto detalhadamente em: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html. Acessado em 25 de fevereiro de 2025.

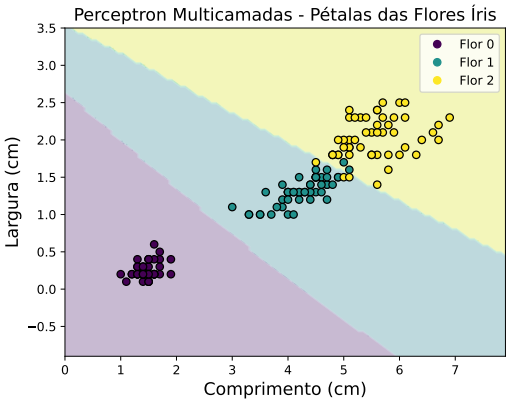
```
# Treinando a rede
pmc.fit(x,d)
# Fazendo previsões
y = pmc.predict(x)
# Avaliando a acurácia
acuracia = accuracy_score(d, y)
# Variável auxiliar 'erro' é uma matriz
erro=d-y
# Reorganização das dimensões da matriz 'erro'
erros_organizados=erro.reshape((3,50))
# Reorganização das dimensões da matriz de saídas y
y_organizados=y.reshape((3,50))
#Apresentação dos dados resultates do treinamento:
#Percentual de acertos do PMC
print(f"Acurácia:{acuracia*100}%")
#Saídas obtidas com os pesos sinápticos treinados
print("Saídas calculadas:\n", y_organizados)
#Diferença entre saídas desejadas e treinadas
print("Erros de categorização\n", erros_organizados)
```

Algoritmo 10 – Treinamento do PMC com o `MLPClassifier` para os dados das flores Íris. Elaborado pelo autor.

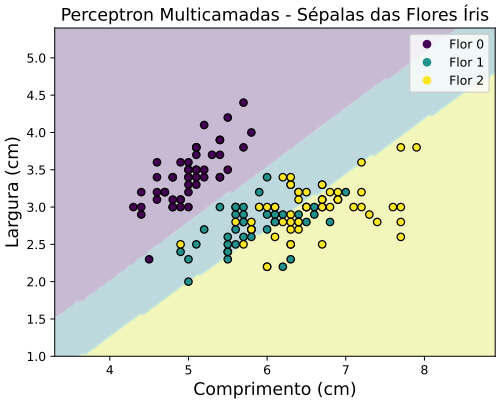
Utilizando somente os dados relativos às medidas de comprimento e largura das pétalas das flores, o PMC como configurado no Algoritmo 9 com as saídas apresentadas pela rede de acordo com o Algoritmo 10, permite que a rede atinja 96% de acurácia. Ao fornecer somente as medidas das sépalas, é alcançada uma acurácia de, aproximadamente, $76,67\%$. Utilizando como amostras as medidas das sépalas e pétalas, a rede apresenta 98.0% de acurácia. Em todos os casos é possível perceber que os equívocos que a rede comete são em relação às classificações das flores Virgínica e Versicolor, que não são linearmente separáveis.

Com os pesos sinápticos treinados, é possível ainda apresentar uma visualização das fronteiras de decisão para cada um dos três casos, como apresentado na Figura 17.

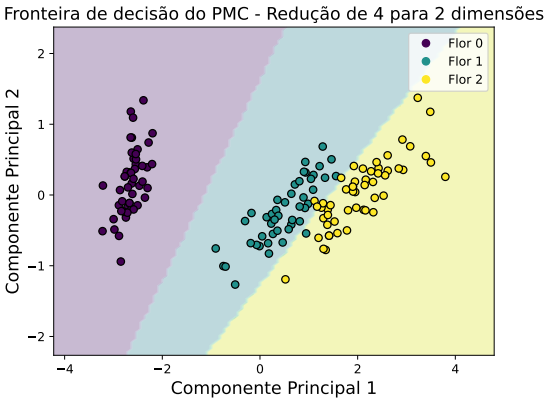
As vantagens em se utilizar somente as pétalas ou sépalas residem no fato dos dados poderem ser dispostos diretamente num plano e a visualização destes ser interpretada diretamente com o modo em que são apresentados, como pode ser visto nas Figuras 17a e 17b.



(a) PMC treinado somente com as pétalas.



(b) PMC treinado somente com as sépalas.



(c) PMC treinado com as pétalas e sépalas.

Figura 17 – PMC treinado com os dados das flores Íris. Elaborados pelo autor.

Ao se utilizar todos os dados de todas as amostras, perde-se o poder de visualização devido ao número de dimensões associada a eles. Em uma proposta para representar os dados em duas dimensões, é possível utilizar como auxílio da Análise dos Componentes Principais (ACP), um recurso que também é fornecido pela biblioteca `scikit-learn`[8] e utilizado para visualização de dados multidimensionais em dimensões reduzidas, como pode ser visto na Figura 17c.

As duas classes e a biblioteca utilizada nestes exemplos não são as únicas possibilidades para implementar um PMC em Python. Além da implementação manual, existem outras bibliotecas como o `Keras` ou o `PyTorch`, que cada uma com seu modo, permitem outras implementações dos parâmetros de configuração do PMC.

[8] "Principal Component Analysis". https://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_iris.html#sphx-glr-auto-examples-decomposition-plot-pca-iris-py. Acessado em 25 de fevereiro de 2025.

Bibliografia

BOTTOU, Léon. **Foreword**. Set. 2017. Perceptrons. Reissue of the 1988 expanded edition. By Marvin L. Minsky and Seymour A. Papert. MIT Press. Cambridge, MA. Disponível em: http://leon.bottou.org/papers/bottou-foreword-2017. Acesso em: 20 nov. 2023.

FISHER, R. A. **Iris**. 1988. DOI: 10.24432/C56C76. Disponível em: https://archive.ics.uci.edu/dataset/53/iris. Acesso em: 25 mar. 2024.

GÉRON, Aurélien. **Mãos à Obra Aprendizado de Máquina com Scikit-Learn & TensorFlow**: Conceitos, Ferramentas e Técnicas Para a Construção de Sistemas Inteligentes. 4. ed. Rio de Janeiro: Alta Books, 2019. P. 576.

HAYKIN, Simon. **Redes Neurais: Princípios e Prática**. Bookman Editora, 2001. ISBN 9788577800865.

KOVÁCS, Zsolt László. **Redes Neurais Artificiais: Fundamentos e Aplicações**. 4. ed.: Livraria da Física, 2023. P. 176.

MINSKY, Marvin; PAPERT, Seymour Aubrey. **Perceptrons: An Introduction to Computational Geometry**. Expanded: The MIT Press, 1988.

OLIVEIRA, Victor Rodrigues de. **Descobrindo Redes Neurais Artificiais**: Minicurso sobre os modelos: Perceptron, Adaline e Perceptron Multicamadas. Disponível em: https://oliveiravictor2.github.io/Minicurso_RNAs/. Acesso em: 22 jan. 2025.

PEDREGOSA, F. et al. **Scikit-learn: Machine Learning in Python**. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

SILVA, Ivan Nunes da; SPATTI, Danilo Hernane; FLAUZINO, Rogério Andrade. **Redes Neurais Artificiais Para Engenharia e Ciências Aplicadas**: Fundamentos Teóricos e Aspectos Práticos. 2. ed. São Paulo: Artliber, 2016. P. 431.

WIDROW, Bernard; LEHR, Michael A. **30 years of adaptive neural networks: perceptron, Madaline, and backpropagation**. **Proceedings of the IEEE**, v. 78, n. 9, p. 1415–1442, 1990. DOI: 10.1109/5.58323.