Perceptron Logístico e Regra Delta Generalizada

Prof. Dr. Guilherme de Alencar Barreto

Junho/2025

Departamento de Engenharia de Teleinformática
Programa de Pós-Graduação em Engenharia de Teleinformática (PPGETI)
Universidade Federal do Ceará (UFC), Fortaleza-CE

gbarreto@ufc.br

1 Introdução

Esta seção discute o efeito de se utilizar uma não-linearidade suave na saída do modelo de neurônio da rede Perceptron simples, em vez de uma não-linearidade abrupta, tal como a função sinal. Esta mudança implicará em alterações na regra de aprendizagem que passará a ser chamada de regra delta generalizada.

Esta pequena mudança promove um grande impacto na aplicabilidade da rede Perceptron simples, uma vez que permite sua aplicação também em problemas de regressão. Recorde que antes a rede Perceptron simples era usada apenas em problemas de classificação de padrões. Isto é possível porque a regra de aprendizado será baseado no gradiente do erro quadrático instantâneo, como no modelo Adaline. Assim, o modelo de neurônio logístico a ser discutido nesta nota de aula, unifica os modelos Adaline e Perceptron simples, que possuem origens distintas [1, 2].

O Perceptron simples com função de ativação sigmoidal (i.e. em forma de S) passa a ser chamado doravante de *Perceptron simples Logístico* ou simplesmente de Perceptron logístico (PL). Aqui iremos discutir duas funções sigmoidais muito utilizadas nos modelos atuais de redes neurais artificiais, a saber, a função sigmoide logística e a função tangente hiperbólica. Porém, pode-se usar qualquer função não-linear suave, que seja diferenciável até segunda ordem, pelo menos.

Manteremos a mesma notação para o vetor de entrada ($\mathbf{x} \in \mathbb{R}^{(p+1)\times 1}$), ativação do i-ésimo neurônio ($u_k \in \mathbb{R}$), saída do k-ésimo neurônio ($y_k \in (0, +1)$ ou $y_k \in (-1, +1)$), peso sináptico conectando a j-ésima entrada ao i-ésimo neurônio ($w_{kj} \in \mathbb{R}$), vetor de pesos do k-ésimo neurônio ($\mathbf{w}_k \in \mathbb{R}^{(p+1)\times 1}$) e número de neurônios (q), melhor definidos a seguir.

O vetor de entrada da rede PL é então definido como

$$\mathbf{x}(t) = \begin{bmatrix} x_0(t) \\ x_1(t) \\ \vdots \\ x_j(t) \\ \vdots \\ x_p(t) \end{bmatrix}_{(p+1)\times 1} = \begin{bmatrix} -1 \\ x_1(t) \\ \vdots \\ x_j(t) \\ \vdots \\ x_p(t) \end{bmatrix}_{(p+1)\times 1} \tag{1}$$

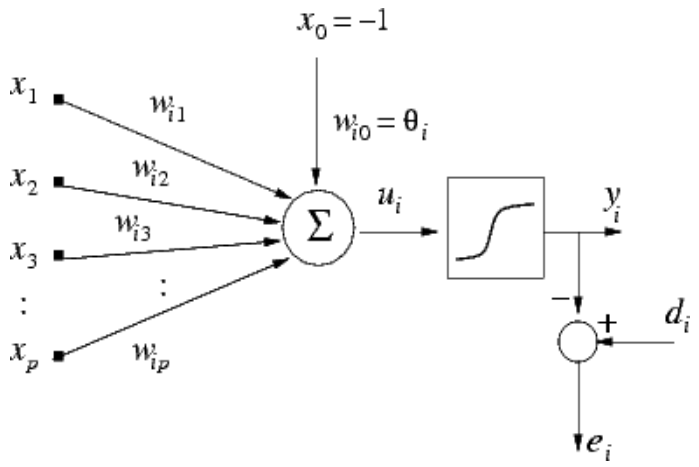


Figura 1: Arquitetura do i-ésimo neurônio da rede Perceptron logístico.

em que $x_j(t)$ denota uma componente qualquer do vetor de entrada $\mathbf{x}(t)$ e t indica o instante de apresentação deste vetor à rede.

O vetor de pesos associado ao i-ésimo neurônio é representado da seguinte forma:

$$\mathbf{w}_k(t) = \begin{bmatrix} w_{k0}(t) \\ w_{k1}(t) \\ \vdots \\ w_{kj}(t) \\ \vdots \\ w_{kc}(t) \end{bmatrix}_{(p+1)\times 1} = \begin{bmatrix} \theta_k(t) \\ w_{k1}(t) \\ \vdots \\ w_{kj}(t) \\ \vdots \\ w_{kc}(t) \end{bmatrix}_{(p+1)\times 1}, \tag{2}$$

em que w_{kj} é o peso sináptico que conecta a entrada j ao k-ésimo neurônio de saída, θ_k define um limiar (*threshold* ou *bias*) associado ao k-ésimo neurônio de saída.

Por fim, o vetor de saídas desejadas (ou saídas-alvo) é representado por um vetor de q componentes, ou seja

$$\mathbf{d}(t) = \begin{bmatrix} d_1(t) \\ \vdots \\ d_k(t) \\ \vdots \\ d_c(t) \end{bmatrix}_{c\times 1}, \tag{3}$$

em que $d_k(t)$ a saída desejada para o k-ésimo neurônio, $k = 1, \ldots, c$.

Importante: Em uma rede PL, cada neurônio possui o seu próprio vetor de pesos $\mathbf{w}_k$. Assim, uma rede com q neurônios terá $p \times q$ pesos sinápticos w_{ij} e q limiares θ_k, resultando em um total de $(p+1) \times q$ parâmetros ajustáveis. Estes parâmetros deverão ser ajustados por meio de uma regra de atualização recursiva denominada **Regra Delta Generalizada**.

2 Regra Delta Generalizada

A ativação do k-ésimo neurônio da rede PL é calculada como segue

$$\begin{aligned} u_k(t) &= \sum_{j=1}^{p} w_{kj}(t)x_j(t) - \theta_k(t) \\ &= \sum_{j=1}^{p} w_{kj}(t)x_j(t) + w_{k0}(t)x_0(t) \\ &= \sum_{j=0}^{p} w_{kj}(t)x_j(t) \\ &= \mathbf{w}_k^T(t)\mathbf{x}(t) = \mathbf{x}^T(t)\mathbf{w}_k(t) \end{aligned} \tag{4}$$

em que foi feito $x_0 = -1$ e $w_{k0} = \theta_k$. O sobrescrito T indica a operação de transposição dos vetores.

Como agora existe uma não-linearidade suave na saída do neurônio, conforme ilustrado na Figura 1, então a sua saída passa a ser representada como

$$y_k(t) = \phi(u_k(t)) = \phi\left(\mathbf{w}_k^T(t)\mathbf{x}(t)\right) = \phi_k(t), \tag{5}$$

em que $\phi(\cdot)$ é uma não-linearidade do tipo sigmoidal, possuindo a forma da letra S esticada. Dois tipos são comumente usados, a sigmoide logística

$$y_k(t) = \phi_k(t) = \frac{1}{1 + \exp\{-u_k(t)\}}, \tag{6}$$

e a tangente hiperbólica

$$y_k(t) = \phi_k(t) = \frac{1 - \exp\{-u_k(t)\}}{1 + \exp\{-u_k(t)\}}. \tag{7}$$

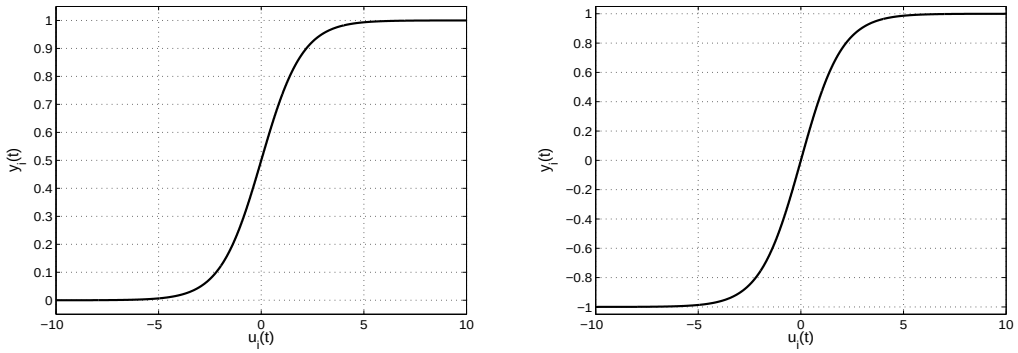
O domínio tanto da função sigmoide logística, quanto da tangente hiperbólica é a reta dos números reais. Contudo, a imagem da função sigmoide logística está restrita ao intervalo $(0, 1)$, enquanto a da tangente hiperbólica está restrita ao intervalo $(-1, +1)$. De um extremo a outro a função é sempre crescente. O gráfico dessas duas funções estão mostrados na Figura 2.

As versões vetoriais das Eqs. (4) e (5) são dadas por

$$\mathbf{u}(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \\ \vdots \\ u_c(t) \end{bmatrix} = \mathbf{W}\mathbf{x}(t) \tag{8}$$

em que $\mathbf{W} \in \mathbb{R}^{c\times(p+1)}$ é a matriz de pesos da rede PL. Nesta matriz, os vetores de pesos $\mathbf{w}_k$, $k = 1, \ldots, q$, estão organizados como linhas de uma matriz $\mathbf{W}$, ou seja

$$\mathbf{W} = \begin{bmatrix} — & \mathbf{w}_1^T & — \\ — & \mathbf{w}_2^T & — \\ \vdots & \vdots & \vdots \\ — & \mathbf{w}_c^T & — \end{bmatrix}, \tag{9}$$



(a) (b)

Figura 2: (a) Função sigmoide logística e (b) tangente hiperbólica.

em que $\dim(\mathbf{W}) = c \times (p+1)$. O vetor com as saídas dos q neurônios da rede é dada por

$$\mathbf{y}(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_c(t) \end{bmatrix} = \begin{bmatrix} \phi(u_1(t)) \\ \phi(u_2(t)) \\ \vdots \\ \phi(u_c(t)) \end{bmatrix} = \begin{bmatrix} \phi(\mathbf{w}_1^T \mathbf{x}(t)) \\ \phi(\mathbf{w}_2^T \mathbf{x}(t)) \\ \vdots \\ \phi(\mathbf{w}_c^T \mathbf{x}(t)) \end{bmatrix} \tag{10}$$

O desempenho do k-ésimo neurônio da rede PL no instante t, como aproximador do mapeamento $\mathbf{F}(\cdot)$, é avaliado a partir do erro quadrático instantâneo definido como

$$J_k(t) = \frac{1}{2} e_k^2(t), \tag{11}$$

$$= \frac{1}{2} \left(d_k(t) - y_k(t) \right)^2, \tag{12}$$

$$= \frac{1}{2} \left(d_k(t) - \phi(u_k(t)) \right)^2, \tag{13}$$

$$= \frac{1}{2} \left(d_k(t) - \phi(\mathbf{w}_k^T(t) \mathbf{x}(t)) \right)^2, \tag{14}$$

em que $e_k(t) = d_k(t) - y_k(t)$ é o erro do i-ésimo neurônio em resposta à apresentação do par entrada-saída atual $(\mathbf{x}(t), \mathbf{d}(t))$.

Assim, define-se a seguir uma medida global do desempenho da rede PL, chamada de **erro quadrático médio** (EQM), com base nos erros produzidos para todos os pares entrada-saída $\{\mathbf{x}(t), \mathbf{d}(t)\}$:

$$J[\mathbf{W}] = \frac{1}{2N} \sum_{t=1}^{N} \sum_{k=1}^{c} J_k(t) = \frac{1}{2N} \sum_{t=1}^{N} \sum_{k=1}^{c} e_k^2(t) = \frac{1}{2N} \sum_{k=1}^{c} \sum_{t=1}^{N} \left[d_k(t) - y_k(t) \right]^2 \tag{15}$$

em que $\mathbf{W}$ denota o conjunto de todos os parâmetros ajustáveis do modelo (pesos e limiares).

Os parâmetros da rede PL devem ser especificados a fim de que os neurônios produzam sempre uma saída $y_k(t)$ bem próxima da saída esperada $d_k(t)$ para um vetor de entrada $\mathbf{x}(t)$ qualquer. Em outras palavras, o funcional $J_k(t)$ deve ter o menor valor possível para aquele conjunto de dados específico. Dá-se o nome de parâmetros ótimos aos valores de $\mathbf{w}_k$ que alcançam este objetivo.

Um procedimento iterativo para se chegar aos parâmetros ótimos envolve o uso do método de otimização baseada no gradiente descendente:

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) - \alpha \frac{\partial J_k(t)}{\partial \mathbf{w}_k(t)} \tag{16}$$

tal que a variável $0 < \alpha < 1$ é a taxa de aprendizagem. Utilizando a regra da cadeia, a derivada presente na Eq. (16) pode ser escrita da seguinte forma:

$$\frac{\partial J_k(t)}{\partial \mathbf{w}_k(t)} = \frac{dJ_k(t)}{de_k(t)} \frac{de_k(t)}{dy_k(t)} \frac{dy_k(t)}{du_k(t)} \frac{\partial u_k(t)}{\partial \mathbf{w}_k(t)} \tag{17}$$

em que os resultados das quatro derivadas para a rede PL são mostrados a seguir:

$$\frac{dJ(t)}{de_k(t)} = e_k(t) \tag{18}$$

$$\frac{de_k(t)}{dy_k(t)} = -1 \tag{19}$$

$$\frac{dy_k(t)}{du_k(t)} = \frac{d\phi(u_k(t))}{du_k(t)} = \phi_k'(t) \tag{20}$$

$$\frac{\partial u_k(t)}{\partial \mathbf{w}_k(t)} = \mathbf{x}(t) \tag{21}$$

Assim, temos que a regra recursiva de ajuste dos pesos, $w_j(t)$, é dada por

$$\begin{aligned} \mathbf{w}_k(t+1) &= \mathbf{w}_k(t) - \alpha \frac{\partial J_k(t)}{\partial \mathbf{w}_k(t)}, \\ &= \mathbf{w}_k(t) + \alpha e_k(t) \phi_k'(t) \mathbf{x}(t), \\ &= \mathbf{w}_k(t) + \alpha \delta_k(t) \mathbf{x}(t), \end{aligned} \tag{22}$$

em que $\delta_k(t)$ é chamado de gradiente local do k-ésimo neurônio. A regra de aprendizagem mostrada na Eq. (22) é conhecida como *regra delta generalizada* ou *regra LMS generalizada*, para diferenciar da regra LMS original que é também conhecida como regra delta. Em todo caso, se a função de ativação $\phi(u_k)(t)$ for a função identidade, ou seja, $\phi(u_k)(t) = u_k(t)$, teremos a regra LMS usual.

A derivada instantânea[1] da função de ativação $\phi_k(t)$, denotada por $\phi_k'(t)$, para a função logística é dada por

$$\phi_k'(t) = \frac{d\phi_k(t)}{du_k(t)} = \frac{d(1 + \exp\{-u_k(t)\})^{-1}}{du_k(t)}, \tag{23}$$

$$= (-1) \cdot (1 + \exp\{-u_k(t)\})^{-2} \cdot \frac{d \exp\{-u_k(t)\}}{du_k(t)}, \tag{24}$$

$$= (-1) \cdot (-1) \cdot (1 + \exp\{-u_k(t)\})^{-2} \cdot \exp\{-u_k(t)\}, \tag{25}$$

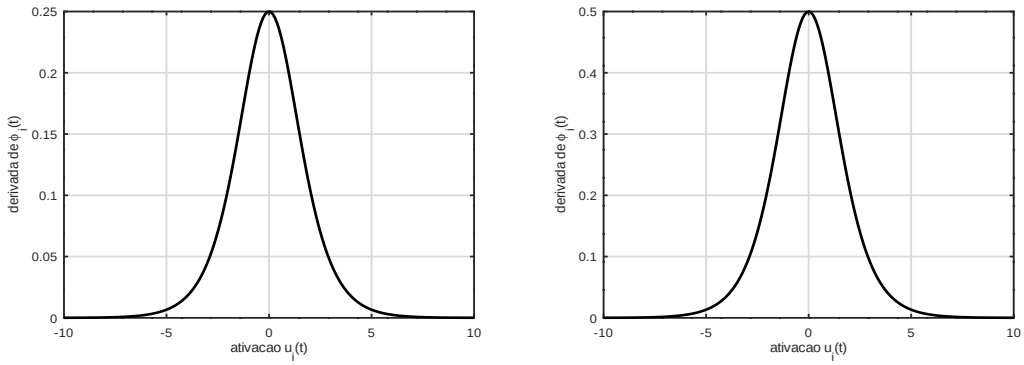
$$= \frac{\exp\{-u_k(t)\}}{(1 + \exp\{-u_k(t)\})^2} = \frac{1}{1 + \exp\{-u_k(t)\}} \cdot \frac{\exp\{-u_k(t)\}}{1 + \exp\{-u_k(t)\}}, \tag{26}$$

$$= \phi_k(t)[1 - \phi_k(t)] = y_k(t)[1 - y_k(t)], \tag{27}$$

e para a função tangente hiperbólica (exercício!) temos que

$$\phi_k'(t) = \frac{1}{2}(1 - \phi_k^2(t)) = \frac{1}{2}(1 - y_k^2(t)). \tag{28}$$

[1]Derivada no instante atual t.



(a) (b)

Figura 3: (a) Função sigmoide logística e (b) tangente hiperbólica.

Uma das vantagens em se utilizar as funções sigmoide logística e tangente hiperbólica reside no fato de suas derivadas instantâneas não precisam ser computadas via métodos numéricos, já que podem ser escritas em função do valor da saída $y_k(t)$ no instante t. Os gráficos das derivadas das funções de ativação sigmoide logística e tangente hiperbólica estão mostrados na Figura 3.

Além disso, a derivada da função de ativação é usada no cálculo do gradiente local do erro do k-ésimo neurônio, $\delta_k(t)$, que pode ser entendido como elemento modulador do erro $e_k(t)$. Assim, valores altos do erro (positivos ou negativos) gerarão valores pequenos da derivada $\phi'_k(t)$, o que confere certa estabilidade ao aprendizado da rede PL ao longo do tempo. Tal estabilidade é verificada por uma curva de aprendizagem (convergência do EQM) mais suave que a da regra LMS convencional. Porém, essa modulação do erro pode também causar o problema conhecido como *paralisia da rede*, em que os pesos não são modificados mesmo para erros grandes, principalmente no começo do processo de aprendizado. Para mitigar a ocorrência de paralisia, recomenda-se iniciar os pesos com valores bem pequenos.

A rede PL pode ser utilizada tanto para aproximação de funções, quanto para classificação de padrões.

3 Versão Matricial da Regra LMS Generalizada

Organizando os gradientes locais dos c neurônios de saída no instante t em um vetor de gradientes locais da seguinte forma:

$$\boldsymbol{\delta}(t) = \begin{bmatrix} \delta_1(t) \\ \delta_2(t) \\ \vdots \\ \delta_c(t) \end{bmatrix}_{c\times 1} \tag{29}$$

podemos escrever a regra de aprendizagem da rede PL da seguinte maneira:

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \alpha\boldsymbol{\delta}(t)\mathbf{x}^T(t), \tag{30}$$

em que esta expressão atualiza a matriz de pesos $\mathbf{W}$, de dimensões $c \times (p+1)$, de uma só vez a cada instante t. Cada termo da Eq. (30) deve ter dimensões compatíveis, de modo que para isso ser possível o termo de correção envolve o produto externo do vetor de erros $\mathbf{e}(t)$ com o vetor de entrada $\mathbf{x}(t)$. É por isso que o vetor de entrada aparece transposto nessa equação.

A versão matricial da regra de aprendizado da rede PL não é muito comum de se ver em livros, porém é extremamente vantajosa para implementação em Octave/Matlab, R e Python, devido à maior velocidade que confere à execução do algoritmo. Usando a Eq. (30) a matriz de pesos $\mathbf{W}$ é atualizada de uma vez a cada apresentação de um vetor de entrada. Note que isto só é possível porque o funcionamento de cada neurônio é independente do funcionamento dos demais, dada a natureza local da regra desta regra de aprendizado.

4 Outras Funções de Ativação

As funções sigmoidais descritas anteriormente ainda são muito usadas em modelos clássicos de redes neurais. Porém, em modelos de redes neurais convolucionais, que com frequência assumem uma escala muito grande, tem se dado preferência há outras funções de menor custo computacional. Descreveremos algumas delas a seguir.

4.1 Rectified Linear Unit (ReLU)

A função de ativação ReLU é uma opção muito popular, que produz a entrada diretamente se ela for positiva e zero caso contrário. Matematicamente, tem-se

$$y_k(t) = \phi(u_k(t)) = \max(0, u_k(t)) = \frac{u_k(t) + |u_k(t)|}{2}. \tag{31}$$

O gráfico da função ReLU no intervalo $(-10, 10)$ para uma grade de valores de u com incremento 0,1 está mostrado na Figura 4a. Algumas propriedades da função de ativação ReLU são lis-adas abaixo.

- Domínio: $u_k \in (-\infty, +\infty)$. Imagem: $y_k(t) \in [0, \infty)$.
- Usada como padrão na maioria da camadas ocultas de redes convolucionais profundas por ser computacionalmente leve e por ajudar a mitigar o problema do desvanecimento (ou enfraquecimento) do gradiente (*vanishing gradient problem*) em redes multicamadas.
- A derivada desta função não satura. Matematicamente, tem-se

$$\phi'_k(t) = \frac{d\phi(u_k(t))}{du_k(t)} = \begin{cases} 1, & u_k(t) > 0 \\ 0, & u_k(t) \leq 0 \end{cases} \tag{32}$$

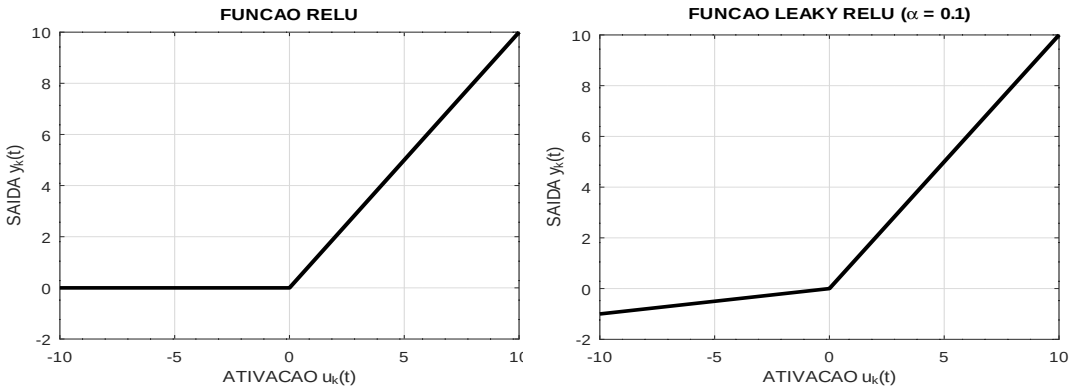
- Produz ativação esparsa; ou seja, muitos neurônios são desativados; ou seja, produzem saída nula, o que melhora a eficiência computacional da rede.

Apesar de possuir muitas vantagens, ela também possui algumas desvantagens, sendo a principal o problema da **ReLU moribunda** (*dying ReLU*). Se muitos neurônios produzirem saídas nulas (especialmente para entradas negativas), eles podem "morrer" durante o treinamento e não serem mais serem ativados.

4.2 Leaky ReLU

Trata-se de uma variação da função ReLU que permite um gradiente pequeno e diferente de zero para entradas negativas, mitigando potencialmente o problema da ReLU moribunda. Matematicamente, tem-se a seguinte expressão para esta função de ativação:

$$y_k(t) = \begin{cases} u_k(t), & u_k(t) \geq 0 \\ \alpha u_k(t), & u_k(t) < 0 \end{cases} \tag{33}$$



(a) (b)

Figura 4: Funções de ativação: (a) ReLU e (b) Leaky ReLU.

em que $0 < \alpha \ll 1$ é uma constante muito pequena, assumindo geralmente o valor $\alpha = 0{,}01$. O gráfico da função Leaky ReLU no intervalo $(-10, 10)$ para uma grade de valores de u com incremento 0,1 está mostrado na Figura 4b.

Algumas propriedades da função de ativação *leaky* ReLU são listadas abaixo:

- Domínio: $u_k \in (-\infty, +\infty)$. Imagem: $y_k(t) \in (-\infty, +\infty)$.
- É computacionalmente leve e ajuda a mitigar o problema da ReLU moribunda.
- Também ajuda a mitigar o problema de neurônios desativados (neurônios mortos).
- A pequena inclinação para valores negativos de ativação pode tornar o treinamento e, consequentemente, a convergência da rede mais lenta.
- A derivada desta função é dada por

$$\phi_k'(t) = \frac{d\phi(u_k(t))}{du_k(t)} = \begin{cases} 1, & u_k(t) \geq 0 \\ \alpha, & u_k(t) < 0 \end{cases} \tag{34}$$

4.3 Exponential Linear Unit (ELU)

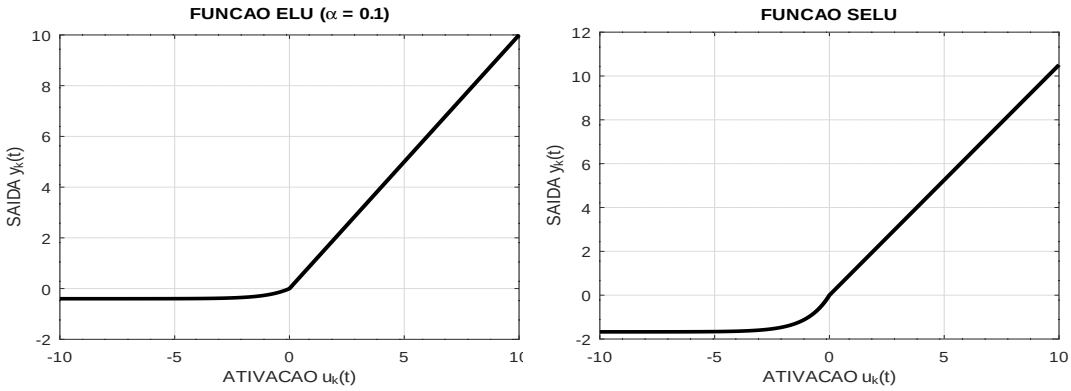
Esta função de ativação se assemelha à *Leaky ReLU*, mas com uma curva suave para ativações negativas, o que pode levar a um aprendizado mais rápido. Matematicamente, tem-se a seguinte expressão para esta função de ativação:

$$y_k(t) = \begin{cases} u_k(t), & u_k(t) \geq 0 \\ \alpha\left(e^{u_k(t)} - 1\right), & u_k(t) < 0 \end{cases} \tag{35}$$

A saída da ELU para valores positivos da ativação $u_k(t)$ é a própria ativação; ou seja, é uma função identidade para $u_k(t) \geq 0$. Se a entrada for negativa, a curva de saída será ligeiramente suavizada em direção à constante α. Quanto maior for esta constante, mais negativa será a saída para entradas negativas. O gráfico da função ELU no intervalo $(-10, 10)$ para uma grade de valores de u com incremento 0,1 está mostrado na Figura 5a.

Algumas propriedades da função de ativação ELU são listadas abaixo:

- Domínio: $u_k \in (-\infty, +\infty)$. Imagem: $y_k(t) \in (-\alpha, +\infty)$.



(a) (b)

Figura 5: Funções de ativação: (a) ELU e (b) SeLU.

- Tende a induzir uma convergência mais rápida da rede que a função ReLU.
- Frequentemente conduz a um melhor desempenho de generalização que a função ReLU.
- É contínua e diferenciável para todo o domínio da função.
- Mitiga o problema do desvanecimento do gradiente.
- Mitiga o problema da explosão do gradiente.
- Mitiga o problema da ReLU moribunda.
- A derivada desta função é dada por

$$\phi_k'(t) = \frac{d\phi(u_k(t))}{du_k(t)} = \begin{cases} 1, & u_k(t) \geq 0 \\ \alpha e^{u_k(t)}, & u_k(t) < 0 \end{cases} \tag{36}$$

Como aspecto negativo, possui um custo computacional maior que a ReLU, devido à não linearidade da função exponencial aplicada aos valores de entrada negativos.

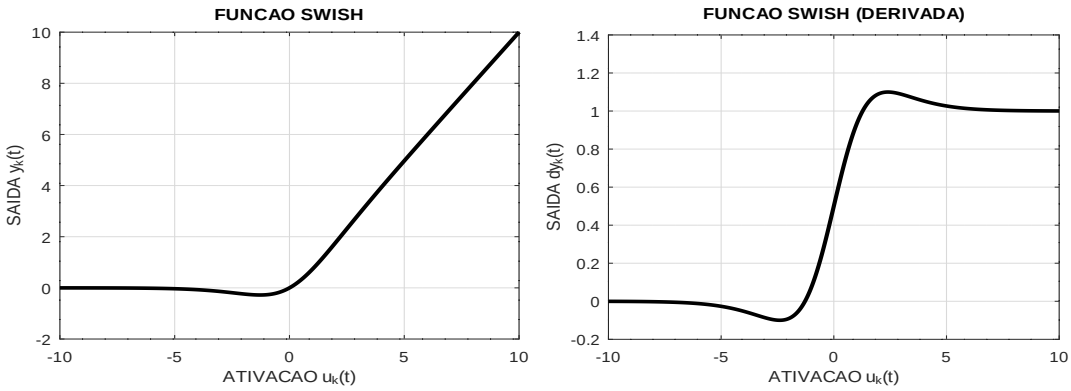
4.4 Scaled Exponential Linear Unit (SELU)

Esta função de ativação foi projetada para lidar com gradientes que desaparecem ou explodem. Matematicamente, ela é muito similar à função ELU:

$$y_k(t) = \begin{cases} \lambda u_k(t), & u_k(t) \geq 0 \\ \alpha \left(e^{u_k(t)} - 1\right), & u_k(t) < 0 \end{cases} \tag{37}$$

em que $\lambda \approx 1{,}0507$ a fim de promover a autonormalização da saída (média = 0, variância = 1) e $\alpha \approx 1{,}67326$. O gráfico da função SELU no intervalo $(-10, 10)$ para uma grade de valores de u com incremento 0,1 está mostrado na Figura 5b.

Basicamente, esta função de ativação mantém as mesmas propriedades da função ELU com o benefício adicional da autonormalização.



(a) (b)

Figura 6: Função de ativação Swish e sua derivada.

4.5 Swish

Trata-se de uma função de ativação que combina a simplicidade da ReLU com algumas vantagens das funções sigmoidais. Matematicamente, tem-se

$$y_k(t) = \phi(u_k(t)) = u_k(t) \cdot \text{logsig}(u_k(t)), \tag{38}$$

em que $logsig(u)$ denota a função de ativação sigmoide logística. O gráfico da função Swish e de sua derivada no intervalo $(-10, 10)$ para uma grade de valores de u com incremento 0,1 está mostrado na Figura 6. Fica como exercício o cálculo matemático da derivada da função Swish.

A função de ativação Swish tem basicamente as mesmas vantagens que as funções ELU e SELU, sendo utilizadas principalmente em redes neurais muito profundas. Esta função foi proposta por pesquisadores da Google como uma alternativa à popular função de ativação ReLU[2].

5 Variações da Regra do Gradiente Descendente

Do mesmo modo pelo qual existem diversas funções de ativação, também existem variantes do método clássico de gradiente descendente estocástico (*stochastic gradient descent*, SGD) aqui instanciado pela Eq. (16). A partir desta equação, chega-se à regra recursiva de ajuste dos parâmetros (pesos e limiares) da rede PL mostrada na Eq. (23).

Nas subseções a seguir apresentaremos 7 variantes da regra LMS generalizada. Em geral, todas estas variantes são extensões bem simples da regra LMS generalizada, não oferecendo muito custo computacional ao processo de atualização dos pesos da rede PL.

5.1 LMS Normalizada

Tomando com o referência a Eq. (16), a regra LMS normalizada é escrita como

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \frac{\alpha}{\gamma + \|\mathbf{x}(t)\|^2}\delta_k(t)\mathbf{x}(t), \tag{39}$$

$$= \mathbf{w}_k(t) + \frac{\alpha}{\gamma + \mathbf{x}(t)^T\mathbf{x}(t)}\delta_k(t)\mathbf{x}(t), \tag{40}$$

[2] `https://arxiv.org/pdf/1710.05941v1`

em que $\|\mathbf{x}\|^2 = \mathbf{x}^T\mathbf{x}$ é a norma euclidiana quadrática do vetor $\mathbf{x}$ e $0 < \gamma \leq 1$ é uma constante usada para evitar divisão por zero. É comum usar $\gamma = 1$.

A regra LMS normalizada promove uma estabilização do aprendizado em função da norma do vetor de entrada $\mathbf{x}(t)$. Se a norma do vetor $\mathbf{x}(t)$ for alta (i.e., $\|\mathbf{x}\|^2 \gg \gamma$), situação comum quando há outliers, a taxa de aprendizado é dividida por $\gamma + \|\mathbf{x}\|^2$, desacelerando o aprendizado, evitando assim que o vetor de pesos seja atualizado de maneira brusca. Se a norma do vetor $\mathbf{x}(t)$ for pequena (i.e., $\|\mathbf{x}\|^2 \ll \gamma$), a taxa de aprendizado é aumentada para α/γ, acelerando o aprendizado sem desestabilizá-lo.

5.2 *Leaky* LMS

A regra leaky LMS é escrita como

$$\mathbf{w}_k(t+1) = (1-\lambda)\mathbf{w}_k(t) + \alpha\delta_k(t)\mathbf{x}(t), \tag{41}$$

em que a constante $0 < \lambda < 1$ é chamada de parâmetro de fuga (*leaky*) ou de decaimento (*decay*).

A regra leaky LMS é popularmente conhecido na área de redes neurais artificiais como regularização por decaimento dos pesos[3] (*weight decay regularization*). No caso em que o vetor de pesos $\mathbf{w}_k$ não é atualizado (i.e., $\delta_k(t) = 0$), seu valor será diminuído de uma fração $-\lambda\mathbf{w}_k(t)$. Isso faz com que o vetor não assuma valores elevados, mantendo a sua norma pequena de modo equivalente à regularização L2, também conhecida como regularização de Tikhonov (ver material sobre regressão via método dos mínimos quadrados ordinários).

Curiosidade: As regras LMS normalizada e leaky podem ser combinadas para gerar uma variante: a regra LMS leaky normalizada.

5.3 LMS com Termo de Momento

A regra LMS com termo de momento é escrita como

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \alpha\delta_k(t)\mathbf{x}(t) + \lambda\Delta\mathbf{w}_k(t-1), \tag{42}$$

em que $0 < \lambda < 1$ é o fator de momento e o termo $\Delta\mathbf{w}_k(t-1)$ é a correção aplicada ao vetor de pesos no passo anterior:

$$\Delta\mathbf{w}_k(t-1) = \alpha\delta_k(t-1)\mathbf{x}(t-1), \tag{43}$$

$$= \mathbf{w}_k(t) - \mathbf{w}_k(t-1). \tag{44}$$

A regra LMS com termo de momento confere estabilidade ao processo de atualização do vetor de pesos $\mathbf{w}_k(t)$ a cada instante t. O termo de momento $\lambda\Delta\mathbf{w}_k(t-1)$ evita que mudanças bruscas ocorram em $\mathbf{w}_k(t)$. Daí a origem do termo *momento*, referindo-se ao conceito físico de *momento de inércia*. Em outras palavras, a introdução do termo $\lambda\Delta\mathbf{w}_k(t-1)$ confere maior *inércia* (ou seja, maior dificuldade de variação) nas correções em $\mathbf{w}_k(t)$.

Um resultado muito interessante sobre essa variante da regra LMS é que para funções quadráticas (e.g., erro médio quadrático), ela é uma versão do método do gradiente conjugado[4].

[3] Anders Krogh & John A Hertz (1992). "A simple weight decay can improve generalization". In: Advances in Neural Information Processing Systems, pp. 950-957.

[4] `http://www.nacad.ufrj.br/~amit/bk03nn.pdf`

5.4 Variantes Adagrad/RMSProp/ADAM

Nesta seção, descreveremos a regra LMS/Adagrad e duas variantes, a saber, a regra RMSProp (*Root Mean Square Propagation*) e a regra ADAM (*Adaptive Moment Estimation*).

O principal conceito por trás da regra LMS/Adagrad é a ideia de ter uma taxa de aprendizado adaptativa, que dependa da soma histórica dos gradientes quadráticos usados na atualização do vetor de pesos[5]. Assim, para cada instante t, o gradiente da função custo (e.g., MSE) com relação ao k-ésimo vetor de pesos é calculado, assim como na regra LMS padrão:

$$\mathbf{g}_k(t) = \frac{\partial J_k(t)}{\partial \mathbf{w}_k(t)}, \tag{45}$$

de modo que o a soma acumulada G_t do quadrado dos gradientes é dada por

$$\begin{aligned} G_k(t) &= G_k(t-1) + \|\mathbf{g}_k(t)\|^2, &(46)\\ &= G_k(t-1) + \left\| \frac{\partial J_k(t)}{\partial \mathbf{w}_k(t)} \right\|^2, &(47)\\ &= G_k(t-1) + \delta_k^2(t)\mathbf{x}^T(t)\mathbf{x}(t). &(48) \end{aligned}$$

Assim, a regra LMS/Adagrad é escrita como

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \frac{\alpha}{\sqrt{\gamma + G_k(t)}}\delta_k(t)\mathbf{x}(t), \tag{49}$$

em que $0 < \gamma < 1$ é uma constante pequena usada para evitar a divisão por zero. O termo $\frac{1}{\sqrt{\gamma+G_k(t)}}$ reduz efetivamente a taxa de aprendizado para os vetores de pesos que têm um gradiente acumulado grande. Em contrapartida, os vetores de pesos com gradientes acumulados menores terão uma taxa de aprendizado maior.

Já a regra RMSProp é uma variante da regra LMS/Adagrad em que a atualização do termo $G_k(t)$ envolve uma combinação convexa dos termos $G_k(t-1)$ e $\|\mathbf{g}_k(t)\|^2$. Matematicamente, tem-se

$$\begin{aligned} G_k(t) &= \lambda G_k(t-1) + (1-\lambda)\|\mathbf{g}_k(t)\|^2, &(50)\\ &= \lambda G_k(t-1) + (1-\lambda)\left\| \frac{\partial J_k(t)}{\partial \mathbf{w}_k(t)} \right\|^2, &(51)\\ &= \lambda G_k(t-1) + (1-\lambda)\delta_k^2(t)\mathbf{x}^T(t)\mathbf{x}(t). &(52) \end{aligned}$$

em que $0 < \lambda < 1$ é o fator de decaimento (tipicamente assume o valor $\lambda = 0{,}9$). Em outras palavras, a regra LMS/RMSProp é como uma regra *leaky* LMS/Adagrad.

Por fim, a regra LMS/ADAM combina os benefícios das regras LMS/Adagrad e LMS com termo de momento ao usar tanto a média móvel dos gradientes, usada para suavizar (ou filtrar) variações bruscas nas componentes do vetor gradiente causadas por ruído estocástico, quanto os gradientes quadráticos para adaptar a taxa de aprendizado[6]. A regra LMS/ADAM necessita das seguintes etapas:

1. **Estimação do 1o. Termo (vetor gradiente filtrado)**:

$$\begin{aligned} \mathbf{m}_k(t+1) &= \beta_1 \mathbf{m}_k(t) + (1-\beta_1)\frac{\partial J_k(t)}{\partial \mathbf{w}_k(t)}, \\ &= \beta_1 \mathbf{m}_k(t) + (1-\beta_1)\delta_k(t)\mathbf{x}(t), &(53) \end{aligned}$$

em que $0 < \beta_1 < 1$ é uma constante (parâmetro de suavização). Valor típico: $\beta_1 = 0{,}9$.

[5] `https://jmlr.org/papers/volume12/duchi11a/duchi11a.pdf`

[6] `https://arxiv.org/abs/1412.6980`

2. **Estimação do 2o. Termo (variância)**:

$$v_k(t+1) = \beta_2 v_k(t) + (1-\beta_2)\left\|\frac{\partial J_k(t)}{\partial \mathbf{w}_k(t)}\right\|^2,$$
$$= \beta_2 v_k(t) + (1-\beta_2)\delta_k^2(t)\mathbf{x}^T(t)\mathbf{x}(t), \quad (54)$$

em que $0 < \beta_2 < 1$ é uma constante. Valor típico: $\beta_2 = 0{,}999$.

3. **Correção de viés para 1o. e 2o. Termos**:

$$\hat{\mathbf{m}}_k(t) = \frac{\mathbf{m}_k(t+1)}{1-\beta_1} \quad \text{e} \quad \hat{v}_k(t) = \frac{v_k(t)}{1-\beta_2} \quad (55)$$

4. **A Regra LMS/ADAM em si:**

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) - \frac{\alpha}{\sqrt{\hat{v}_k(t)} + \gamma}\hat{\mathbf{m}}_k(t), \quad (56)$$

em que $0 < \gamma \leq 1$ é uma constante usada para evitar divisão por zero.

Comentário Final: Não há uma função de ativação ou uma variante da regra LMS que funcione melhor que todas as outras para todas a aplicações. Portanto, é necessário experimentar duas ou mais combinações a fim de chegar na melhor solução para o problema em mãos. Comece sempre pela configuração mais básica, tipo logsig+LMS padrão, em seguida vá testando combinações até encontrar uma solução que satisfaça o problema com o menor custo computacional possível.

6 Exercícios Computacionais

Exercício 1 - Qual é a implicação do aparecimento do fator $\phi_k'(t)$ na regra delta generalizada? Dica: Pesquise por *paralisia da rede* em sítios da internet.

Exercício 2 - Escolher um conjunto de dados no Repositório da Universidade da Califórnia em Irvine (`http://www.ics.uci.edu/~mlearn/MLSummary.html`) e utilizar os pares de vetores entrada-saída correspondentes para avaliar o Perceptron simples Logístico em um problema de classificação de padrões e em um problema de regressão. Determinar a curva de aprendizagem (EQM em função da época de treinamento) e os índices de desempenhos correspondentes a cada tarefa. Classificação: taxas de acerto médio, desvio padrão, mediana, mínimo e máximo. Regressão: R2 médio, desvio padrão, mediana, mínimo e máximo.

Referências

[1] B. Widrow and M. A. Lehr. Perceptrons, adalines, and backpropagation. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 871–877. MIT Press, 2nd edition, 2002.

[2] B. Widrow and M. A. Lehr. 30 years of adaptive neural networks: Perceptron, madaline and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442, 1990.