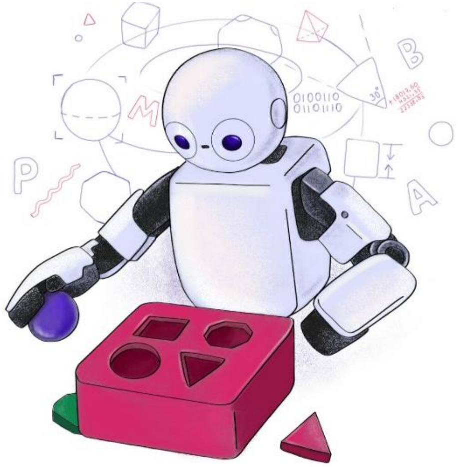
TP555 - Inteligência Artificial e Machine Learning:
Aprendizado em Redes Neurais com Múltiplas Camadas



Inatel

Felipe Augusto Pereira de Figueiredo
felipe.figueiredo@inatel.br

Rede Neural com Múltiplas Camadas

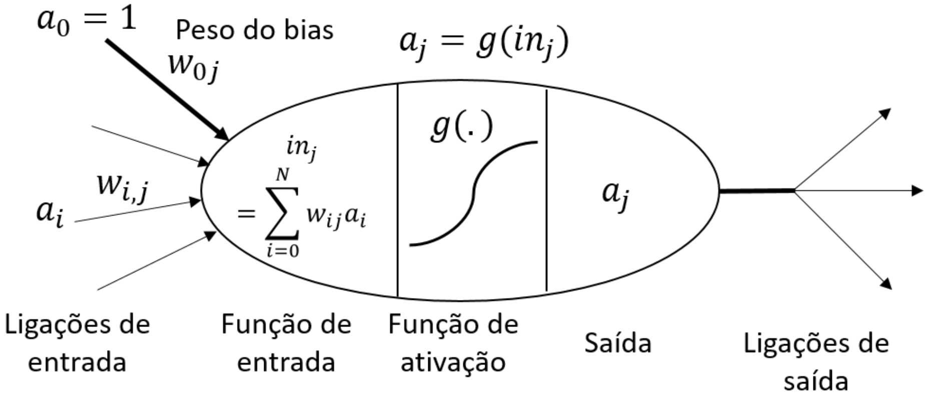
- Uma ***ligação*** do ***nó*** i para o ***nó*** j serve para propagar o sinal de ativação do ***nó*** i para o ***nó*** j. Cada ***ligação*** tem um ***peso*** associado, $w_{i,j}$, que determina a ***força*** e ***sinal*** da ***ligação***.
- Assim como nos modelos da ***regressão linear***, cada ***nó*** tem a entrada 0, a_0, sempre com valor igual a 1 e um peso associado $w_{0,j}$. Ou seja, esta entrada não está conectada a nenhum outro ***nó***.
- Cada ***nó*** j, calcula inicialmente uma soma ponderada de suas entrada da seguinte forma

$$in_j = \sum_{i=0}^{K} w_{i,j} a_i.$$

- Em seguida, o ***nó*** aplica uma ***função de ativação*** (ou de limiar), $g(.)$, ao somatório acima para obter sua saída

$$a_j = g(in_j) = g\left(\sum_{i=0}^{K} w_{i,j} a_i\right) = g(\boldsymbol{w}^T \boldsymbol{a}).$$

- Existem vários tipos de ***funções de ativação***, $f(.)$, que podem ser utilizadas pelos ***nós***.
- Após termos decidido qual função de ativação utilizar, conectamos os neurônios para formar uma rede.
- Os ***nós*** podem ser conectados de forma a se formar redes ***feed-forward*** (conexões em apenas uma direção) ou ***recorrentes*** (saídas se conectam às entradas). Nós focaremos nas redes do tipo ***feed-forward***, onde cada nó recebe como entradas as saídas de todos os nós anteriores.



$$a_j = g\left(\sum_{i=0}^{K} w_{i,j} a_i\right),$$

onde a_i é a saída do nó i e $w_{i,j}$ é o peso conectando a saída do nó i a este nó, o nó j.

Dividir para conquistar

- Considere uma rede neural do tipo ***feed-forward*** com múltiplas camadas.
- Para uma função de custo, ou ***loss***, L_2 ou seja, uma função que use o método dos mínimos quadrados para calcular o erro, então, para qualquer ***peso*** w a derivada da função de custo, $loss(.)$, em relação ao peso w é dada por

$$\frac{\partial}{\partial w} loss(\boldsymbol{w}) = \frac{\partial}{\partial w} |\boldsymbol{y} - \boldsymbol{h}_{\boldsymbol{w}}(\boldsymbol{x})|^2 = \frac{\partial}{\partial w} \sum_k (y_k - a_k)^2 = \sum_k \frac{\partial}{\partial w} (y_k - a_k)^2 ,$$

onde o índice k varia ao longo dos nós da camada de saída, $\boldsymbol{h}_{\boldsymbol{w}}$ e $\boldsymbol{y}$ são vetores e a_k é a saída do k-ésimo nó.

- Cada termo do somatório final é simplesmente o gradiante da função de custo para a k-ésima saída da rede, computado como se as outras saídas não existissem, ou seja, elas são desconsideradas.
- Desta forma, como pode ser percebido, nós podemos decompor um problema de aprendizado de m-saídas em m problemas de aprendizado, desde que nos lembremos de somar as contribuições do gradiente de cada um deles ao atualizar os pesos.

Retropropagação

- A maior complicação vem da adição de camadas escondidas (ou ocultas) à rede.
- Enquanto o erro $\boldsymbol{y} - \boldsymbol{h_w}(\boldsymbol{x})$ na camada de saída é claro, o erro nas camadas escondidas parece algo ***misterioso*** devido ao fato de que os dados de treinamento não dizerem qual valor os nós escondidos devem ter.
- Felizmente, acontece que podemos ***retropropagar*** (do Inglês, back-propagate) o erro da camada de saída para as camadas escondidas.
- O processo de retropropagação surge diretamente da derivação do gradiente geral de erro.
- Inicialmente, vou descrever o processo através de uma justificativa intuitiva, então, em seguida, apresentarei a derivação.

Intuição por trás da retropropagação

- O gradiante da função de custo para a k-ésima saída da rede é dado por

$$\begin{aligned}\frac{\partial}{\partial w}(y_k - a_k)^2 &= 2(y - a_k)\frac{\partial(y - a_k)}{\partial w}\\ &= -2(y - a_k)g'(\boldsymbol{wx})\frac{\partial \boldsymbol{wx}}{\partial w}\\ &= -2(y - a_k)g'(\boldsymbol{wx})x,\end{aligned}$$

onde $g(\boldsymbol{wx}) = a_k$. Para a derivação acima, nós utilizamos a ***regra da cadeia***:

$$\frac{\partial g(f(x))}{\partial x} = \frac{\partial g(f(x))}{\partial f}\frac{\partial f(x)}{\partial x} = g'\big(f(x)\big)\frac{\partial f(x)}{\partial x}.$$

Intuição por trás da retropropagação

- A derivada g' da função logistica satisfaz $g'(z) = g(z)(1 \; - g(z))$, desta forma nós temos
$$g'(\boldsymbol{wx}) = g(\boldsymbol{wx})\big(1 - g(\boldsymbol{wx})\big) = a_k(1 - a_k).$$
- Portanto, a regra de atualização dos pesos para a camada de saída é dada por
$$w \leftarrow w + \alpha(y - a_k)a_k(1 - a_k)x.$$
- Nós temos múltiplos nós de saída, então deixemos E_k ser o k-ésimo componente do vetor de erro $\boldsymbol{y} - \boldsymbol{h_w}(\boldsymbol{x})$.
- Também seráconveniente definirmos um erro modificado como $\Delta_k = E_k g'(in_k)$, de forma que a regra de atualização dos pesos se torna
$$w_{j,k} \leftarrow w_{j,k} + \alpha a_j \Delta_k,$$

onde $in_k = \sum_{i=0}^{n} w_{i,k} a_i$.

Intuição por trás da retropropagação

- Para atualizar as conexões entre os nós de entrada e os nós escondidos, nós precisamos definir uma quantidade análoga ao termo de erro para os nós de saída.
- É aqui onde nós realizamos a ***retropropagação***. A ideia é que o nó escondido j é "responsável" por uma fração do erro Δ_k em cada um dos nós de saída ao qual ele se conecta.
- Portanto, os valores Δ_k são divididos de acordo com a força da conexão entre o nó escondido e o nó de saída e são propagados de volta para fornecer os valores Δ_j para a camada escondida.
- A regra de propagação para os valores Δ é a seguinte:

$$\Delta_j = g'(in_j) \sum_k w_{j,k} \, \Delta_k .$$

Intuição por trás da retropropagação

- Agora, a regra de atualização dos pesos para os pesos entre as entradas e a camada escondida é essencialmente idêntica à regra de atualização para a camada de saída:

$$w_{i,j} \leftarrow w_{i,j} + \alpha a_i \Delta_j.$$

- Assim, o processo de retropropagação pode ser resumido da seguinte forma:
  - Calcule os valores Δ para os nós de saída, utilizando o erro observado.
  - Começando com a camada de saída, repita o seguinte para cada camada da rede, até que a primeira camada oculta seja alcançada:
    - Propague os valores Δ de volta para a camada anterior.
    - Atualize os pesos entre as duas camadas.

Algoritmo detalhado de retropropagação para redes com múltiplas camadas

function BackPropagateLearning(*exemplos*, *rede*) **returns** uma rede neural

inputs: *exemplos*, um conjunto de exemplos, cada um com vetor de entrada x e vetor de saída y.

rede, uma rede multi-camadas com L camadas, pesos $w_{i,j}$, activation function g

local variables: Δ, um vetor de erros, indexado pelo nó da rede

repeat

for each peso $w_{i,j}$ **in** rede **do**

$w_{i,j} \leftarrow$ um número aleatório com valor pequeno

for each exemplo $(\boldsymbol{x}, \boldsymbol{y})$ **in** exemplos **do**

/*propague as entradas para frente para computar as saídas*/

for each nó i **in** camada de entrada **do**

$a_i \leftarrow x_i$

for l=2 **to** L **do**

for each nó j **in** camada l **do**

$in_j \leftarrow \sum_{i=0}^{n} w_{i,j} a_i$

$a_j \leftarrow g(in_j)$

/*propague os valores deltas para trás, partindo da camada de saída para a camada de entrada*/

for each nó j **in** camada de saída **do**

$\Delta[j] \leftarrow g'(in_j)(y_j - a_j)$

for l = L -1 **to** 1 **do**

for each nó i **in** camada l **do**

$\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$

/*atualize cada um dos pesos da rede usando ps valores delta*/

for each peso $w_{i,j}$ **in** rede **do**

$w_{i,j} \leftarrow w_{i,j} + \alpha a_i \Delta[j]$

until algum critério de parad seja satisfeito

return rede

Derivação das equações de retro-propagação

- A derivação é bastante similar à feita para o cálculo do gradiente para o regressor logístico, com exceção que devemos usar a regra da cadeia mais de uma vez.
- Dada a equação $\frac{\partial}{\partial w} loss(\boldsymbol{w}) = \sum_k \frac{\partial}{\partial w}(y_k - a_k)^2$, nós começamos calculando apenas o gradiente para $loss_k = (y_k - a_k)^2$ na k-ésima saída. O gradiente para essa equação de ***custo*** com relação aos pesos conectando a camada escondida à camada de saída será igual a zero exceto para pesos $w_{j,k}$ que se conectam ao k-ésimo nó de saída. Para aqueles pesos, nós temos

$$\frac{\partial loss_k}{\partial w_{j,k}} = -2(y_k - a_k)\frac{\partial a_k}{\partial w_{j,k}} = -2(y_k - a_k)\frac{\partial g(in_k)}{\partial w_{j,k}}$$

$$= -2(y_k - a_k)g'(in_k)\frac{\partial in_k}{\partial w_{j,k}} = -2(y_k - a_k)g'(in_k)\frac{\partial}{\partial w_{j,k}}\left(\sum_j w_{j,k}a_j\right)$$

$$= -2(y_k - a_k)g'(in_k)a_j = -a_j\Delta_k,$$

com Δ_k definido como antes.

Derivação das equações de retro-propagação

- Para obtermos o gradiente com respeito aos pesos $w_{i,j}$ conectando a camada de entrada à camada oculta, nós precisamos expandir as ativações a_j e reaplicar a ***regra da cadeia***.
- Na sequência, veremos a derivação com bastante detalhes com o intuito de observarmos como o operador de derivada parcial propaga de volta pela rede

$$\frac{\partial loss_k}{\partial w_{i,j}} = -2(y_k - a_k)\frac{\partial a_k}{\partial w_{i,j}} = -2(y_k - a_k)\frac{\partial g(in_k)}{\partial w_{i,j}}$$

$$= -2(y_k - a_k)g'(in_k)\frac{\partial in_k}{\partial w_{i,j}} = -2\Delta_k \frac{\partial}{\partial w_{i,j}}\left(\sum_j w_{j,k}a_j\right)$$

$$= -2\Delta_k w_{j,k}\frac{\partial a_j}{\partial w_{i,j}} = -2\Delta_k w_{j,k}\frac{\partial g(in_j)}{\partial w_{i,j}}$$

$$= -2\Delta_k w_{j,k} g'\left(in_j\right)\frac{\partial in_j}{\partial w_{i,j}}$$

$$= -2\Delta_k w_{j,k} g'\left(in_j\right)\frac{\partial}{\partial w_{i,j}}\left(\sum_i w_{i,j}a_i\right)$$

$$= -2\Delta_k w_{j,k} g'\left(in_j\right)a_i = -a_i\Delta_j,$$

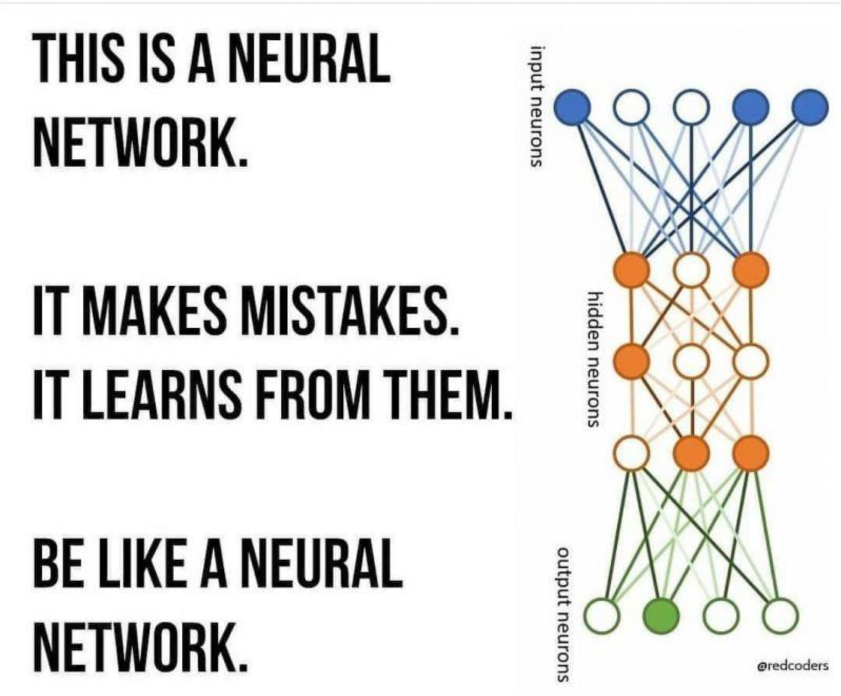
com Δ_j definido como antes.

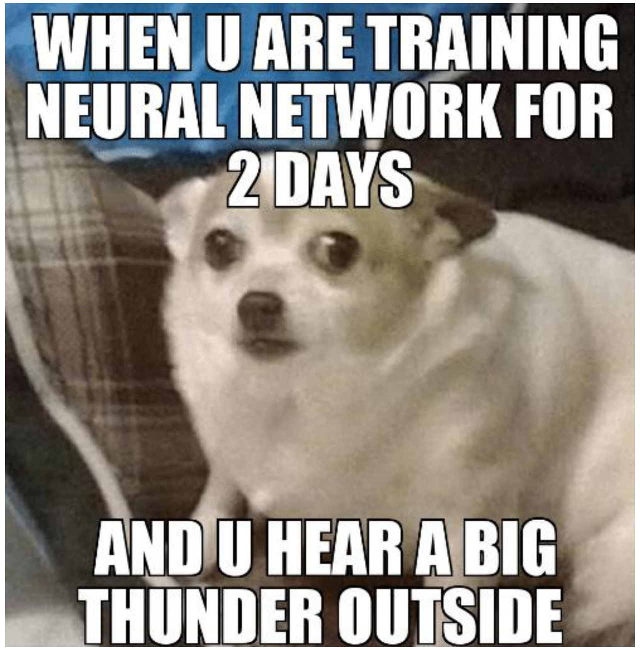
Derivação das equações de retro-propagação

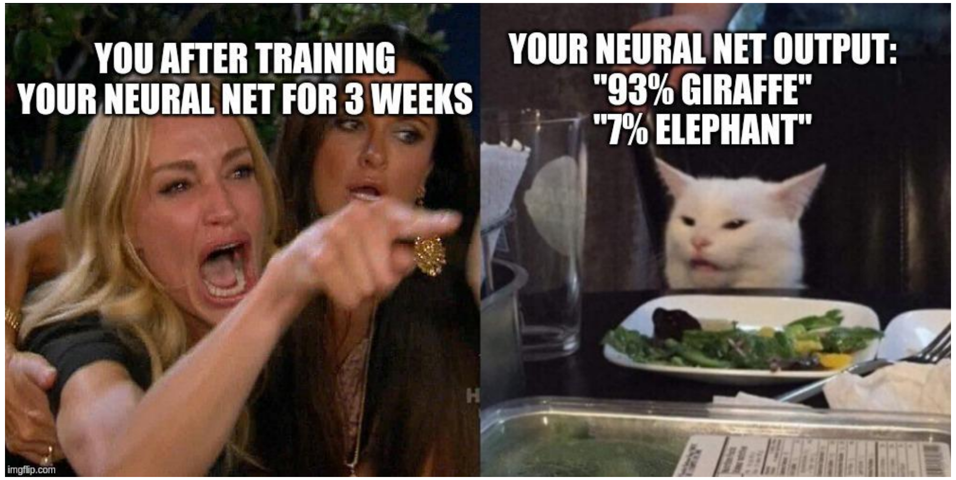
- Através das derivações anteriores, nós obtemos as regras de atualização obtidas anteriormente através da análise intuitiva.
- Fica claro através dos cálculos anteriores que o processo pode se continuado para redes neurais com mais de uma camada escondida, o que justifica o algortimo apresentado anteriormente.

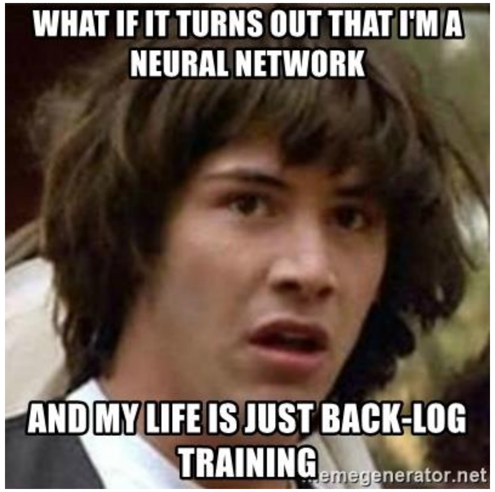
Obrigado!


STILL WAITING
FOR MY NEURAL NETWORK TO TRAIN
memegenerator.net


THIS IS A NEURAL NETWORK.
IT MAKES MISTAKES.
IT LEARNS FROM THEM.
BE LIKE A NEURAL NETWORK.
input neurons
hidden neurons
output neurons
@redcoders

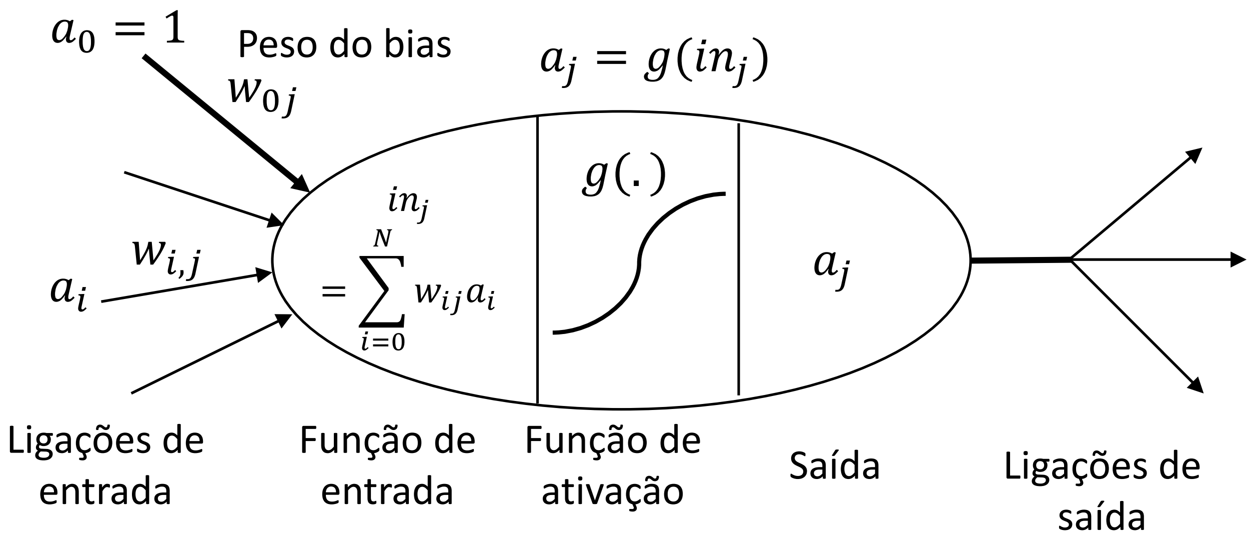

WHEN U ARE TRAINING NEURAL NETWORK FOR 2 DAYS
AND U HEAR A BIG THUNDER OUTSIDE

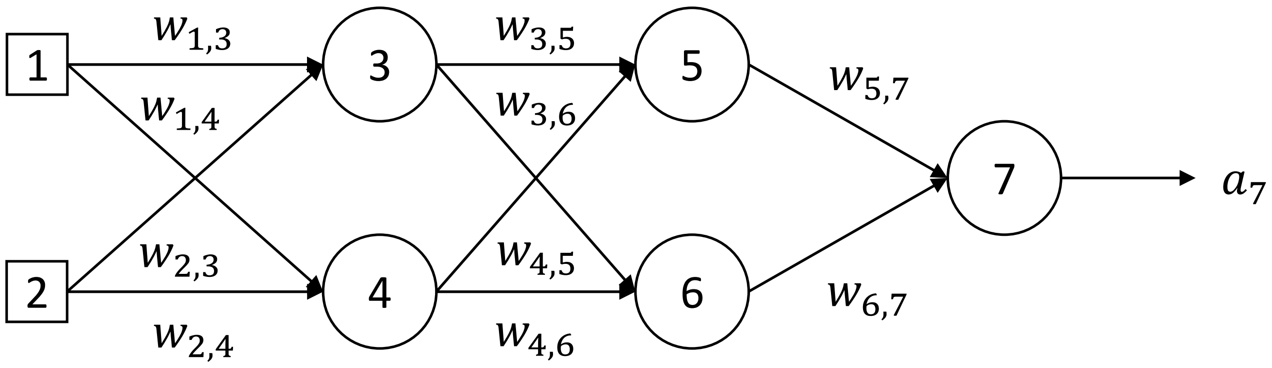

YOU AFTER TRAINING YOUR NEURAL NET FOR 3 WEEKS
YOUR NEURAL NET OUTPUT:
"93% GIRAFFE"
"7% ELEPHANT"
imgflip.com


WHAT IF IT TURNS OUT THAT I'M A NEURAL NETWORK
AND MY LIFE IS JUST BACK-LOG TRAINING
memegenerator.net


SAY DEEP LEARNING
ONE MORE TIME

Figuras


$a_0 = 1$
Peso do bias
w_{0j}
$a_j = g(in_j)$
$g(.)$
in_j
$= \sum_{i=0}^{N} w_{ij} a_i$
$w_{i,j}$
a_i
a_j
Ligações de entrada
Função de entrada
Função de ativação
Saída
Ligações de saída



$w_{1,3}$
$w_{1,4}$
$w_{2,3}$
$w_{2,4}$
$w_{3,5}$
$w_{3,6}$
$w_{4,5}$
$w_{4,6}$
$w_{5,7}$
$w_{6,7}$
1
2
3
4
5
6
7
a_7