

Mais sobre Tensorflow e funções de ativação



Nas aulas anteriores, aprendemos:

- como fazer previsões usando uma rede neural já treinada no NumPy ou no Tensorflow. → **(problema de propagação para frente - inferência)**
- Também aprendemos a treinar um modelo usando Tensorflow

Nas aulas anteriores, aprendemos:

- como fazer previsões usando uma rede neural já treinada no NumPy ou no Tensorflow. → **(problema de propagação para frente - inferência)**
- Também aprendemos a treinar um modelo usando Tensorflow

Pergunta:

Como o Tensorflow faz para encontrar valores adequados para os parâmetros $w_j^{[l]}$, $b^{[l]}$?

Passo 1: Definição do modelo

$$f_{\vec{w}, b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$$

Pergunta: Quais são as características desse modelo?

Passo 1: Definição do modelo

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$$

Pergunta: Quais são as características desse modelo?

Passo 2: Definição da função perda e da função custo

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = -y^{(i)} \log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) - (1 - y^{(i)}) \log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)$$

$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right)$ é a função de **perda logística**, e mede quanto o modelo $f_{\vec{w},b}(\vec{x}^{(i)})$ está errando em relação ao rótulo verdadeiro $y^{(i)}$ para uma dada amostra i . É também chamada de **função de entropia cruzada binária**

$$J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^m L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) \quad \rightarrow \quad \textbf{Função custo: média das perdas}$$

Relembrando da Regressão Logística

Passo 1: Definição do modelo

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$$

Pergunta: Quais são as características desse modelo?

Passo 2: Definição da função perda e da função custo

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = -y^{(i)} \log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) - (1 - y^{(i)}) \log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)$$

$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right)$ é a função de **perda logística**, e mede quanto o modelo $f_{\vec{w},b}(\vec{x}^{(i)})$ está errando em relação ao rótulo verdadeiro $y^{(i)}$ para uma dada amostra i . É também chamada de **função de entropia cruzada binária**

$$J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^m L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) \quad \rightarrow \quad \textbf{Função custo: média das perdas}$$

Passo 3: Minimizar a função custo

Encontrar os valores de \vec{w} , b que minimizam $J(\vec{w}, b)$ pelo Método do Gradiente:

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b) \quad j = 1, \dots, n$$

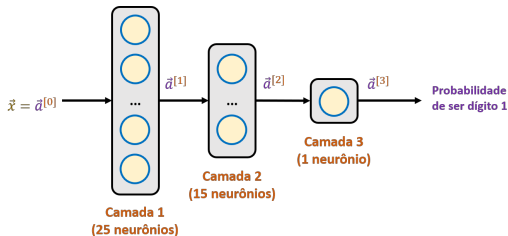
$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w}, b)$$

E para as redes neurais, como fica?

E para as redes neurais, como ficam esses 3 passos?

Passo 1: Definição do modelo

Qual seria o modelo $f_{W,B}(\vec{x})$ para a rede neural abaixo?



$$\vec{a}^{[1]} = \begin{bmatrix} g(\vec{w}_1^{[1]} \cdot \vec{x} + b_1^{[1]}) \\ \vdots \\ g(\vec{w}_{25}^{[1]} \cdot \vec{x} + b_{25}^{[1]}) \end{bmatrix} \rightarrow \vec{a}^{[2]} = \begin{bmatrix} g(\vec{w}_1^{[2]} \cdot \vec{a}^{[1]} + b_1^{[2]}) \\ \vdots \\ g(\vec{w}_{15}^{[2]} \cdot \vec{a}^{[1]} + b_{15}^{[2]}) \end{bmatrix}$$

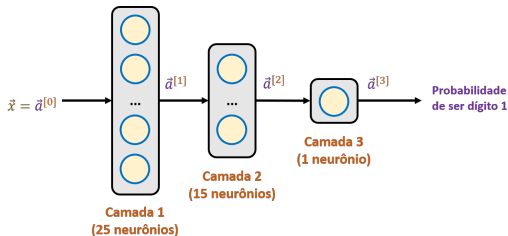
Finalmente:

$$\vec{a}^{[3]} = f_{W,B}(\vec{x}) = g(\vec{w}_1^{[3]} \cdot \vec{a}^{[2]} + b_1^{[3]})$$

OBS: Na representação acima, W, B representam todos os parâmetros da rede neural.

Passo 1: Definição do modelo

Qual seria o modelo $f_{W,B}(\vec{x})$ para a rede neural abaixo?



No Tensorflow, fazemos a definição desse modelo a partir do seguinte código:

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(units=25, activation='sigmoid'),
    Dense(units=15, activation='sigmoid'),
    Dense(units=1, activation='sigmoid'),
```

Passo 2: Definição da função perda e da função custo

$$L\left(f_{W,B}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = -y^{(i)} \log \left(f_{W,B}\left(\vec{x}^{(i)}\right)\right) - (1 - y^{(i)}) \log \left(1 - f_{W,B}\left(\vec{x}^{(i)}\right)\right)$$

Ou seja, é a mesma função perda usada na Regressão Logística (**função de entropia cruzada binária**)

$$J(W, B) = \frac{1}{m} \sum_{i=1}^m L\left(f_{W,B}\left(\vec{x}^{(i)}\right), y^{(i)}\right) \quad \rightarrow \quad \text{Função custo: média das perdas}$$

No Tensorflow, fazemos essa definição compilando o modelo da seguinte forma:

```
model.compile(loss= BinaryCrossentropy())
```

from tensorflow.keras.losses import
BinaryCrossentropy



Passo 2: Definição da função perda e da função custo


$$L\left(f_{W,B}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = -y^{(i)} \log \left(f_{W,B}\left(\vec{x}^{(i)}\right)\right) - (1 - y^{(i)}) \log \left(1 - f_{W,B}\left(\vec{x}^{(i)}\right)\right)$$

Ou seja, é a mesma função perda usada na Regressão Logística (**função de entropia cruzada binária**)

$$J(W, B) = \frac{1}{m} \sum_{i=1}^m L\left(f_{W,B}\left(\vec{x}^{(i)}\right), y^{(i)}\right) \rightarrow \text{Função custo: média das perdas}$$

No Tensorflow, fazemos essa definição compilando o modelo da seguinte forma:

```
model.compile(loss= BinaryCrossentropy())
```

from tensorflow.keras.losses import
BinaryCrossentropy  Keras

Alternativamente, para problemas de regressão, podemos usar:

$$J(W, B) = \frac{1}{m} \sum_{i=1}^m \left(f_{W,B}\left(\vec{x}^{(i)}\right) - y^{(i)}\right)^2 \rightarrow \text{erro quadrático médio}$$

```
model.compile(loss= MeanSquaredError())
```

from tensorflow.keras.losses import
MeanSquaredError

Passo 3: Minimizar a função custo

Encontrar os valores de $w_j^{[l]}$, $b^{[l]}$ que minimizam $J(W, B)$ pelo Método do Gradiente:

$$w_j^{[l]} = w_j^{[l]} - \alpha \frac{\partial}{\partial w_j^{[l]}} J(W, B)$$

$$b^{[l]} = b^{[l]} - \alpha \frac{\partial}{\partial b^{[l]}} J(W, B)$$

Observação:

As derivadas acima podem ser calculadas usando uma metodologia conhecida como **propagação para trás** (*back propagation*)

Passo 3: Minimizar a função custo

Encontrar os valores de $w_j^{[l]}$, $b^{[l]}$ que minimizam $J(W, B)$ pelo Método do Gradiente:

$$w_j^{[l]} = w_j^{[l]} - \alpha \frac{\partial}{\partial w_j^{[l]}} J(W, B)$$

$$b^{[l]} = b^{[l]} - \alpha \frac{\partial}{\partial b^{[l]}} J(W, B)$$

Observação:

As derivadas acima podem ser calculadas usando uma metodologia conhecida como **propagação para trás** (*back propagation*)

No Tensorflow, aplicamos um método alternativo ao método do gradiente padrão usando o comando:

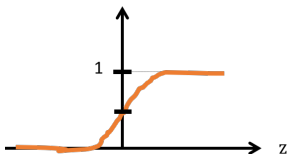
```
model.fit(X, y, epochs=100)
```

Até agora, usamos a função **sigmoide** como função de ativação para todos os neurônios da nossa rede.

Pergunta:

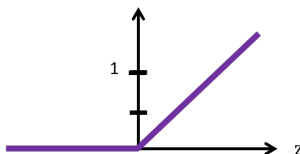
Seria possível utilizar outras funções de ativação? Se sim, como eu faço para escolher a função de ativação mais adequada para cada camada da minha rede?

Conhecendo a função **Unidade Linear Retificada** (ReLU)



Função sigmoide:

$$g(z) = \frac{1}{1+e^{-z}}$$

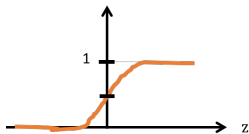


Função ReLU:

$$g(z) = \max(0, z)$$

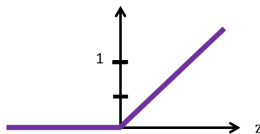
- Assim como a função sigmoide, a função **ReLU** é vastamente utilizada em redes neurais.
- Note que a saída não fica limitada ao intervalo $(0, 1)$. Ao invés disso, ela pode assumir qualquer valor real ≥ 0 .

Conhecendo a função de Ativação Linear



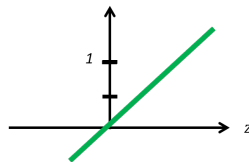
Função **sigmoide**:

$$g(z) = \frac{1}{1+e^{-z}}$$



Função **ReLU**:

$$g(z) = \max(0, z)$$



Função **Linear**:

$$g(z) = z$$

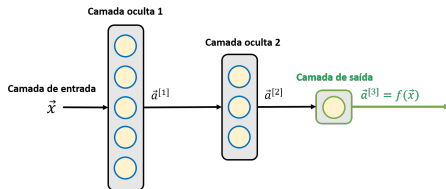
- Como a função de ativação linear é dada por

$$g(z) = z = \vec{w} \cdot \vec{x} + b$$

alguns usuários falarão que usar a essa função específica equivale a não utilizar nenhuma função de ativação.

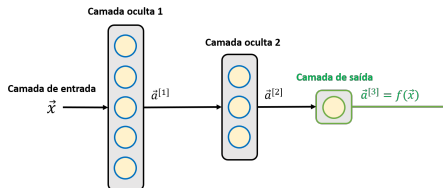
- Essas 3 funções de ativação, e também a função de ativação conhecida como tangente hiperbólica, são as mais utilizadas no contexto de redes neurais

Qual função de ativação devo escolher?



Pensando primeiro na **camada de saída**...

Qual função de ativação devo escolher?

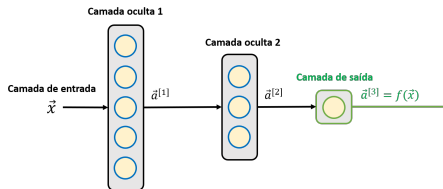


Pensando primeiro na **camada de saída**...

Problemas de classificação

- Se for um problema onde a variável alvo y é 0 ou 1 (problema de classificação binária), então quase sempre a função **sigmoide** será a escolha mais adequada.
- Assim, a saída da nossa rede $f(\vec{x})$ estará no intervalo (0,1) e pode ser interpretada como sendo a probabilidade de y ser 1.

Qual função de ativação devo escolher?



Pensando primeiro na **camada de saída**...

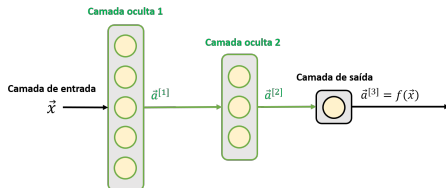
Problemas de classificação

- Se for um problema onde a variável alvo y é 0 ou 1 (problema de classificação binária), então quase sempre a função **sigmoide** será a escolha mais adequada.
- Assim, a saída da nossa rede $f(\vec{x})$ estará no intervalo (0,1) e pode ser interpretada como sendo a probabilidade de y ser 1.

Problemas de regressão

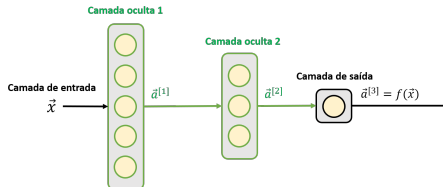
- Se for um problema de regressão onde y pode assumir qualquer valor real positivo ou negativo, então podemos usar função **linear**.
- Se for um problema de regressão onde y pode assumir apenas valores reais positivos (≥ 0), então podemos usar a função **ReLU**.
- Em ambos os casos, a saída da nossa rede $f(\vec{x})$ pode ser interpretada como sendo a nossa estimativa para o y .

Qual função de ativação devo escolher?



Pensando agora nas **camadas ocultas**...

Qual função de ativação devo escolher?



Pensando agora nas **camadas ocultas**...

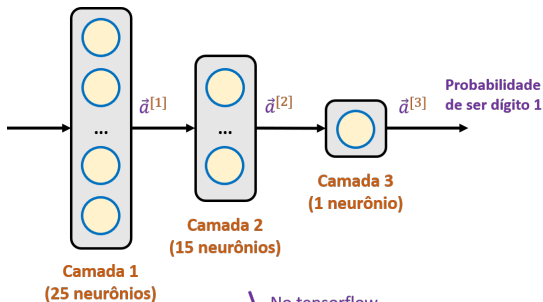
- Hoje em dia, a função de ativação **ReLU** é a mais utilizada nas camadas ocultas.
- Isso porque ela pode ser calculada mais rapidamente (não requer o cálculo de uma exponencial).
- Com a sua definição $\max(0, z)$, a função ReLU possui a capacidade de desativar (zerar) características numa rede, acionando-as apenas quando se faz necessário!

Qual função de ativação devo escolher?

Observação final sobre funções de ativação

Há também outros tipos de funções de ativação. Fique à vontade para pesquisar outras possíveis funções na Internet...

Exemplo no Tensorflow: reconhecimento de dígitos 0 e 1



No tensorflow...

```
modelo = Sequential(  
    [  
        Dense(units=25, activation="relu"),  
        Dense(units=15, activation="relu"),  
        Dense(1, activation="sigmoid")  
    ]  
)  
modelo.compile(  
    loss=BinaryCrossentropy()  
)  
modelo.fit(  
    X,y,epochs=50  
)
```

Constrói o modelo

Define função de perda

Treina modelo

Seja o sistema elétrico de potência abaixo, dividido em 2 áreas (Área Interna e Área Externa)

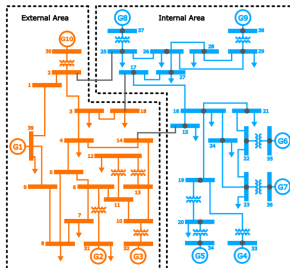


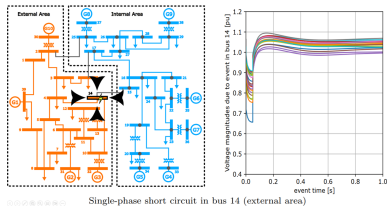
Fig. 1 39-bus New England IEEE benchmark system divided into internal area and external area. Gray circles shown above the buses of the internal area indicate the installation of PMUs.

Supondo que tem uma falta (curto-circuito monofásico) ocorrendo nesse sistema, como localizar se esse evento está ocorrendo na Área Externa ($y = 0$) ou na Área Interna ($y = 1$)?

De olho no código!

No nosso modelo, podemos usar como características \vec{x} os sinais elétricos adquiridos por PMUs instaladas em diferentes pontos do sistema.

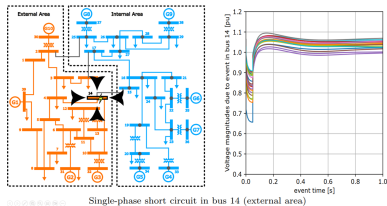
Veja abaixo um exemplo de medição feita por PMUs para um curto-circuito na Área Externa.



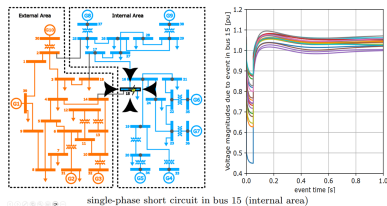
De olho no código!

No nosso modelo, podemos usar como características \vec{x} os sinais elétricos adquiridos por PMUs instaladas em diferentes pontos do sistema.

Veja abaixo um exemplo de medição feita por PMUs para um curto-circuito na Área Externa.



Por outro lado, veja abaixo um exemplo de medição feita por PMUs para um curto-circuito na Área Interna.



De olho no código!

Acesse o Python Notebook usando o QR code ou o link abaixo:

https://colab.research.google.com/github/xaximpvp2/master/blob/main/codigo_aula17_localizacao_falta_SEP.ipynb



Clique nos links abaixo para baixar os dados necessários para rodar o código:

- https://ufprbr0-my.sharepoint.com/:x:/g/personal/ricardo_schumacher_ufpr_br/EaouvKXaSBtDg5ezTEyMApEBHFQ01bGav75PyfdpLZ-BHw?e=eXD1KA
- https://ufprbr0-my.sharepoint.com/:x:/g/personal/ricardo_schumacher_ufpr_br/ESGKrOrZIHJF11rftuMol5EB0oLt8TTOUJk2DBaEWxuKWA?e=vkjrr7
- https://ufprbr0-my.sharepoint.com/:x:/g/personal/ricardo_schumacher_ufpr_br/EZN1REORoRVHhAS3RCemp1YBpHR61m0255fsvaR5Wmt_rg?e=70tNxg
- https://ufprbr0-my.sharepoint.com/:x:/g/personal/ricardo_schumacher_ufpr_br/Ed4U8TZVq9hGo2D7FHYB-8B_7r-TCCFCjc3qAWhH-ykgQ?e=gcsu9l

OBS: Para adicionar os dados ao ambiente do Colab Notebook, no menu do canto esquerdo da tela do Colab clique em "Arquivos" e depois "Fazer upload para o armazenamento da sessão". Então carregue os arquivos baixados.

Parte 1

Rode todo o código. Responda às questões nele contidas e complete-o, se necessário.

- Por que resultados ligeiramente diferentes são obtidos cada vez que o código é rodado?

Parte 2

- 1 Testando diferentes valores para o número de camadas da rede e para o número de neurônios em cada camada, busque uma topologia de rede neural que resulta numa taxa de acerto para os dados de validação $> 80\%$