Inferência usando Redes Neurais (*forward propagation*)

Tensorflow versus NumPy

Na aula anterior, usamos o Tensorflow como um facilitador para resolvermos o problema de inferência.

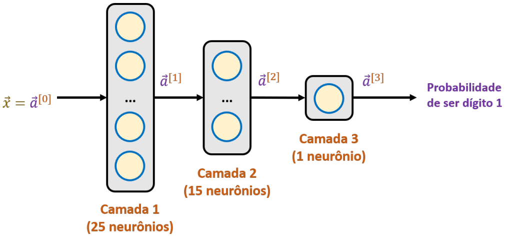
Pergunta:

Se você tivesse que implementar a **propagação para frente** do zero em Python, sem usar Tensorflow, como você faria?

OBS: Alguém criou o ambiente Tensorflow, assim como o ambiente PyTorch. E se você quisesse criar um ambiente equivalente?

Exemplo: Reconhecendo dígitos 0 e 1 escritos à mão

Em uma aula anterior nossa, vimos o seguinte exemplo:



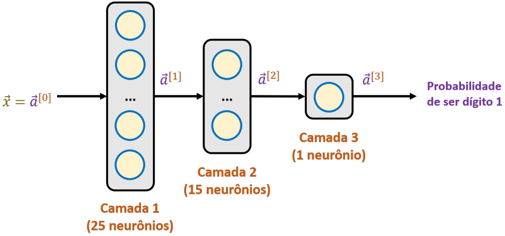
Para a Camada 1, temos:

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

Para a Camada 2, temos:

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

Em uma aula anterior nossa, vimos o seguinte exemplo:



Finalmente, para a Camada 3, temos:

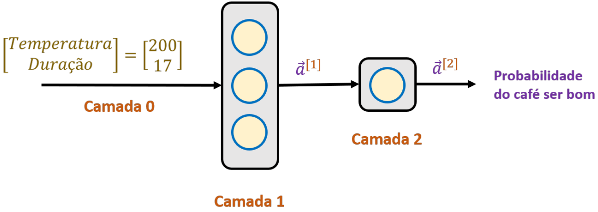
$$\vec{a}^{[3]} = g(\vec{w}_1^{[3]} \cdot \vec{a}^{[2]} + b_1^{[3]})$$

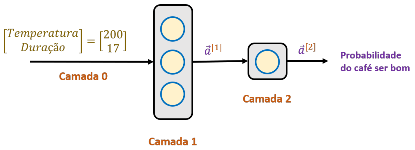
se $a^{[3]} \geq 0.5$, $\hat{y} = 1$ (imagem contém dígito 1).

se $a^{[3]} < 0.5$, $\hat{y} = 0$ (imagem contém dígito 0).

Exemplo: torrefação de café

Como ficaria para o exemplo de Torrefação de café abaixo?





Para a Camada 1, temos:

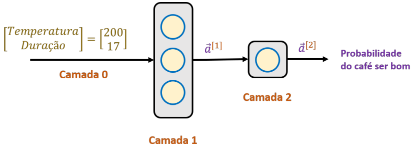
$$\vec{a}^{[1]} = \begin{bmatrix} g(\vec{w}_1^{[1]} \cdot \vec{x} + b_1^{[1]}) \\ g(\vec{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]}) \\ g(\vec{w}_3^{[1]} \cdot \vec{x} + b_3^{[1]}) \end{bmatrix}$$

Para a Camada 2, temos:

$$\vec{a}^{[2]} = g(\vec{w}_1^{[2]} \cdot \vec{a}^{[1]} + b_1^{[2]})$$

se $a^{[2]} \geq 0.5$, $\hat{y} = 1$ (café bom).

se $a^{[2]} < 0.5$, $\hat{y} = 0$ (café ruim).



Pergunta: O que cada linha de código abaixo faz?

```
x = np.array([200, 17])
```

```
w1_1 = np.array([1, 2])
b1_1 = np.array([-1])
z1_1 = np.dot(w1_1,x)+b1_1
a1_1 = sigmoid(z1_1)
```

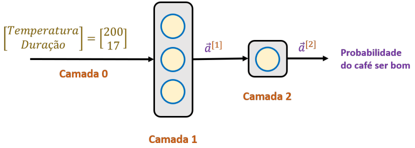
```
w1_2 = np.array([-3, 4])
b1_2 = np.array([1])
z1_2 = np.dot(w1_2,x)+b1_2
a1_2 = sigmoid(z1_2)
```

```
w1_3 = np.array([5, -6])
b1_3 = np.array([-1])
z1_3 = np.dot(w1_3,x)+b1_3
a1_3 = sigmoid(z1_3)
```

```
a1 = np.array([a1_1, a1_2, a1_3])
```

```
w2_1 = np.array([1, 2, 3])
b2_1 = np.array([1])
z2_1 = np.dot(w2_1,a1)+b2_1
a2_1 = sigmoid(z2_1)
```

Exemplo: torrefação de café



Generalizando o conceito de propagação para frente:

```
x = np.array([200, 17])
```

```
W1 = np.array([[ 1, -3,  5],
               [ 2,  4, -6]])
b1 = np.array([-1, 1, -1])
W2 = np.array([[1],[2],[3]])
b2 = np.array([1])
```

Estoca todos os parâmetros da camada 1

Estoca todos os parâmetros da camada 2

```
def dense(a_in,W,b):
    unidades  = W.shape[1]
    a_out     = np.zeros(unidades)
    for j in range(unidades):
        w        = W[:,j]
        z        = np.dot(w,a_in) + b[j]
        a_out[j] = sigmoid(z)
    return a_out
```

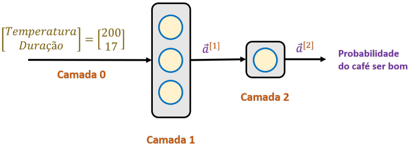
Definindo função que calcula saída de uma dense = camada

Retornando saída calculada da camada

```
def sequential(x):
    a1  = dense(x,W1,b1)
    a2  = dense(a1,W2,b2)
    f_x = a2
    return f_x
```

Calculando sequencialmente as saídas das camadas

Exemplo: torrefação de café



Implementação equivalente usando Vetorização (operações matriciais):

```python
X = np.array([[200, 17]])
```

Aqui, todas as variáveis são matrizes, ou seja, **arrays 2D**

```python
W1 = np.array([[1, -3, 5],
               [2, 4, -6]])
B1 = np.array([[-1, 1, -1]])
W2 = np.array([[1],[2],[3]])
B2 = np.array([[1]])
```

```python
def minha_camada_dense_vetorizada(A_in,W,B):
    Z     = np.matmul(A_in,W)+B
    A_out = sigmoid(Z)
    return A_out
```

Como funciona essa multiplicação de matrizes?

```python
def sequencial(X):
    A1  = minha_camada_dense_vetorizada(X,W1,B1)
    A2  = minha_camada_dense_vetorizada(A1,W2,B2)
    F_X = A2
    return F_X
```

Sejam os dois vetores abaixo

$$\overrightarrow{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad \text{e} \quad \overrightarrow{w} = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

Calcule:

- O produto escalar $\overrightarrow{a} \cdot \overrightarrow{w}$
- O transposto de $\overrightarrow{a}$, ou seja, $\overrightarrow{a}^T$
- O transposto de $\overrightarrow{w}$, ou seja, $\overrightarrow{w}^T$
- A multiplicação vetor-vetor $\overrightarrow{a}^T \overrightarrow{w}$

Sejam as duas matrizes abaixo:

$$A = \begin{bmatrix} 200 & 17 \end{bmatrix} \quad \text{e} \quad W = \begin{bmatrix} 1 & -3 & 5 \\ 2 & 4 & -6 \end{bmatrix}$$

Calcule:

- A é um vetor ou uma matriz?
- Qual é o número de linhas e de colunas da matriz A? E da matriz W?
- É possível realizar a multiplicação matriz-matriz AW?
- Seja a multiplicação $C = AW$. Qual será a dimensão de C?
- Quanto vale $C = AW$?

Calcule $f(\overrightarrow{x})$ para os dois códigos abaixo

```python
x = np.array([200, 17])

W1 = np.array([[ 1, -3,  5],
               [ 2,  4, -6]])
b1 = np.array([-1, 1, -1])
W2 = np.array([[1],[2],[3]])
b2 = np.array([1])

def dense(a_in,W,b):
    unidades = W.shape[1]
    a_out    = np.zeros(unidades)
    for j in range(unidades):
        w        = W[:,j]
        z        = np.dot(w,a_in) + b[j]
        a_out[j] = sigmoid(z)
    return a_out

def sequential(x):
    a1  = dense(x,W1,b1)
    a2  = dense(a1,W2,b2)
    f_x = a2
    return f_x
```

versus

```python
X = np.array([[200, 17]])

W1 = np.array([[1, -3, 5],
               [2, 4, -6]])
B1 = np.array([[-1, 1, -1]])
W2 = np.array([[1],[2],[3]])
B2 = np.array([[1]])

def minha_camada_dense_vetorizada(A_in,W,B):
    Z     = np.matmul(A_in,W)+B
    A_out = sigmoid(Z)
    return A_out

def sequencial(X):
    A1  = minha_camada_dense_vetorizada(X,W1,B1)
    A2  = minha_camada_dense_vetorizada(A1,W2,B2)
    F_X = A2
    return F_X
```

Resposta:

Sejam as três matrizes abaixo:

$$X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad \text{e} \quad b = \begin{bmatrix} -1 & -1 \end{bmatrix}$$

Pergunta:

- É possível somar $X + b$?

Observação:

- O NumPy possui uma capacidade de **broadcasting** incrível. Entretanto, devemos ficar atentos para verificar se estamos entendendo corretamente a operação que está sendo de fato realizada.

De olho no código!

De olho no código!

Usando Numpy, no código a seguir iremos implementar a propagação para frente para o problema de reconhecimento de dígitos escritos à mão.
Entretanto, o código se inicia usando o Tensorflow para encontrar parâmetros adequados para o modelo.

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

`https://colab.research.google.com/github/xaximpvp2/master/blob/main/codigo_aula_16_reconhecendo_digitos_0_e_1_escritos_a_mao.ipynb`

Clique no link abaixo para baixar os dados necessários para rodar o código:

`https://ufprbr0-my.sharepoint.com/:f:/g/personal/ricardo_schumacher_ufpr_br/EvZlARkJilxEpaj_S4kjksgBQDohnTWgZxqKRd3qxrE9Pw?e=dEZ6az`

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.

Parte 2

1. Inclua no código um comando que calcula a taxa de acerto da rede neural. Qual foi a taxa de acerto obtida? Quantas amostras o modelo classificou erroneamente?

OBS: Lembre-se do comando (np.mean(Y == ychapeu) * 100) utilizado na atividade de programação anterior.