Implementando uma Rede Neural do Zero usando NumPy

Adaptado por: Dr. Arnaldo de Carvalho Junior, Março 2026

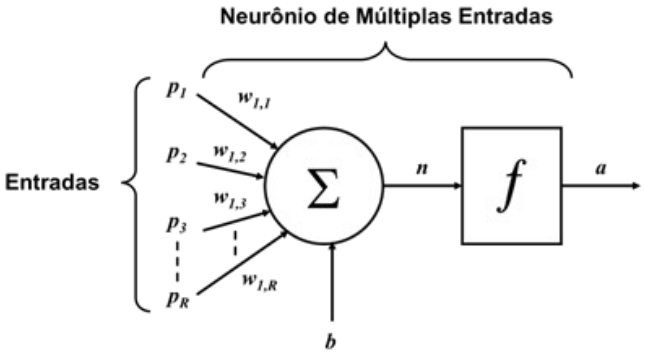
INTRODUÇÃO

As redes neurais artificiais (*artificial neural networks* – ANN) são um componente central dos modelos de aprendizagem profunda (*deep learning* - DL), e implementá-las do zero é uma ótima forma de entender seu funcionamento interno. [1].

Uma rede neural é um modelo computacional inspirado na forma como as redes neurais biológicas processam informações. Consiste em camadas de nós interconectados, chamados neurônios, que transformam dados de entrada em saída. Uma rede neural típica consiste em:

- Camada de entrada (X1, X2, X3) : Toma as características dos dados como entrada.
- Camadas Ocultas (Z): Camadas entre a entrada e a saída que realizam transformações com base em pesos.
- Camada Saída: Produz a previsão final.

A Figura 1 apresenta um neurônio de múltiplas entradas e suas equações básicas [2].



(a)

$$n = w_{1,1}p_1 + w_{1,2}p_2 + w_{1,3}p_3 + \cdots + w_{1,R}p_R$$

(b)

Figura 1 – Neurônio de múltiplas entradas (a) e equações (b).

Várias funções podem servir de função de ativação (*f*), tais como sigmoide, tangente hiperbólica, retificadora linear (ReLU) e suas variações, lógica paraconsistente anotada de dois valores (PAL2v), entre outras (2].

Várias são as topologias de redes neurais, sendo as mais comuns a feedforward, rede neural recorrente e rede neural convolucional [3]. Neste tutorial será demonstrado como implementar um algoritmo básico de ANN do zero usando a bilblioteca NumPy em Python, com foco na construção de um classificador de três letras para os caracteres A, B e C [1].

IMPLEMENTAÇÃO PYTHON

1. Criando o conjunto de dados usando matrizes NumPy de 0 e 1

Como a imagem é uma coleção de valores de pixels em matriz, criaremos um conjunto de dados simples para as letras A, B e C usando matrizes binárias. Essas matrizes representam valores de pixel de grades 5x6 para cada letra, conforme a Figura 2 [1].

Figura 2 – Matrizes

```
# Criando conjunto de dados

# A
a =[0, 0, 1, 1, 0, 0,
   0, 1, 0, 0, 1, 0,
   1, 1, 1, 1, 1, 1,
   1, 0, 0, 0, 0, 1,
   1, 0, 0, 0, 0, 1]
# B
b =[0, 1, 1, 1, 1, 0,
   0, 1, 0, 0, 1, 0,
   0, 1, 1, 1, 1, 0,
   0, 1, 0, 0, 1, 0,
   0, 1, 1, 1, 1, 0]
# C
c =[0, 1, 1, 1, 1, 0,
   0, 1, 0, 0, 0, 0,
   0, 1, 0, 0, 0, 0,
   0, 1, 0, 0, 0, 0,
   0, 1, 1, 1, 1, 0]

# Criando rótulos
sim =[[1, 0, 0],
   [0, 1, 0],
   [0, 0, 1]]
```

representando as letras A, B e C.

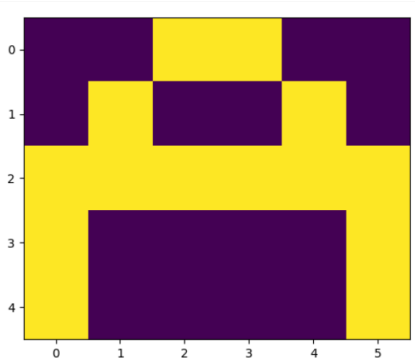
2. Visualizando o conjunto de dados

Para visualizar os conjuntos de dados, pode-se usar Matplotlib para traçar as imagens de cada letra. Isso dará uma compreensão clara de como são os dados antes de alimentá-los na rede neural, conforme a Figura 3 [1].

```python
import numpy as np
import matplotlib.pyplot as plt

# visualizing the data, plotting A.
plt.imshow(np.array(a).reshape(5, 6))
plt.show()
```

(a)



(b)

Figura 3 – Biblioteca Matplotlib para visualização das letras (a) e saída da letra A (b).

3. Como o conjunto de dados está na forma de lista, pode-se convertê-lo em um array.

Converte-se as listas de valores de pixel e os rótulos correspondentes em Matrizes NumPy para trabalhar com eles de forma eficiente na ANN, conforme a Figura 4.

```python
# converting data and labels into numpy array
x =[np.array(a).reshape(1, 30), np.array(b).reshape(1, 30),
                                np.array(c).reshape(1, 30)]
y = np.array(y)
# Printing data and labels
print(x, "\n\n", y)
```

(a)

```
[array([[0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1]])
<TAG1>,
 array([[0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0]])
<TAG1>,
 array([[0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0]])

[[1 0 0]
[0 1 0]
[0 0 1]]
```

(b)

Figura 4 – Código para conversão de dados em matriz (a) e impressão de dads e etiquetas de saída (b)

4. Definindo a Arquitetura da ANN

Neste tutorial, a rede neural terá a seguinte estrutura, conforme a Figura 5:

- Camada de entrada: 1 camada com 30 nós (representando a grade 5x6).
- Camada Oculta: 1 camada com 5 nós.
- Camada Saída: 1 camada com 3 nós (representando as letras A, B e C).

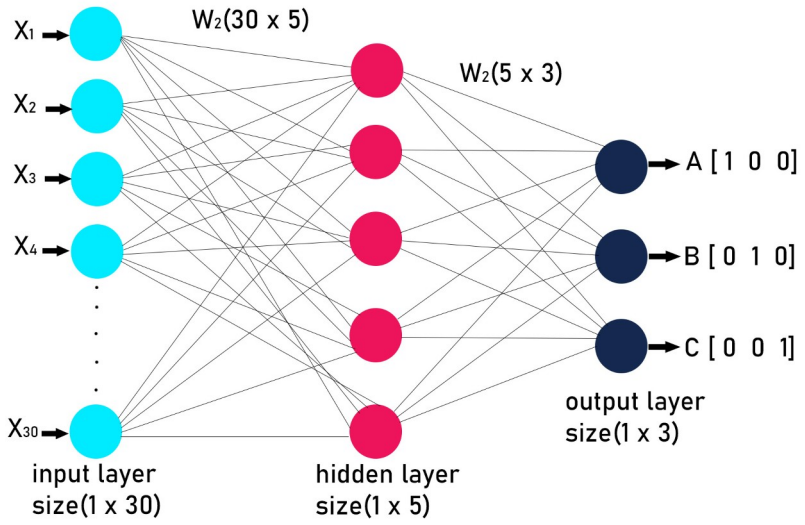


Figura 5 – ANN utilizada neste tutorial.

5. Definindo as funções da rede neural

Aqui, definem-se os principais componentes da rede neural, conforme Figura 6:

- Função Ativação: Será usada a função de ativação sigmóide [2].
- Processo *Feedforward*: Calcula a saída passando a entrada pelas camadas.

- Retropropagação (*backorioagation*): Atualiza pesos para minimizar a perda.
- Função Perda: Será usado o Erro quadrático médio (*mean square error* - MSE) para calcular a perda [4],[5].

```python
# activation function
def sigmoid(x):
    return(1/(1 + np.exp(-x)))

# Creating the Feed forward neural network
def f_forward(x, w1, w2):
    # hidden
    z1 = x.dot(w1)    # input from layer 1
    a1 = sigmoid(z1)  # out put of layer 2
    z2 = a1.dot(w2)   # input of out layer
    a2 = sigmoid(z2)  # output of out layer
    return(a2)

# initializing the weights randomly
def generate_wt(x, y):
    li =[]
    for i in range(x * y):
        li.append(np.random.randn())
    return(np.array(li).reshape(x, y))

# for loss we will be using mean square error(MSE)
def loss(out, Y):
    s =(np.square(out-Y))
    s = np.sum(s)/len(y)
    return(s)

# Back propagation of error
def back_prop(x, y, w1, w2, alpha):

    # hidden layer
    z1 = x.dot(w1)
    a1 = sigmoid(z1)
    z2 = a1.dot(w2)
    a2 = sigmoid(z2)

    # error in output layer
    d2 =(a2-y)
    d1 = np.multiply((w2.dot((d2.transpose()))).transpose(),
                                (np.multiply(a1, 1-a1)))
    # Gradient for w1 and w2
    w1_adj = x.transpose().dot(d1)
    w2_adj = a1.transpose().dot(d2)

    # Updating parameters
    w1 = w1-(alpha*(w1_adj))
    w2 = w2-(alpha*(w2_adj))

    return(w1, w2)
```

Figura 6 – Parâmetros da ANN.

6. Inicializando Pesos

O próximo passo é inicializar os pesos tanto para a camada oculta quanto para a camada de saída aleatoriamente, conforme Figura 7.

```
w1 = generate_wt(30, 5)
w2 = generate_wt(5, 3)

print(w1, "\n\n", w2)
```

(a)

(b)

```
[[ 0,75696605 -0,15959223 -1,43034587 0,17885107 -0,75859483]
 [-0,22870119 1,05882236 -0,15880572 0,11692122 0,58621482]
 [ 0,13926738 0,72963505 0,36050426 0,79866465 -0,17471235]
 [ 1,00708386 0,68803291 0,14110839 -0,7162728 0,69990794]
 [-0,904371310,63977434 -0,433172120,67134205 -0,9316605 ]
 [ 0,15860963 -1,17967773 -0,70747245 0,22870289 0,00940404]
 [ 1.40511247 -1.29543461 1.41613069 -0.97964787 -2.86220777]
 [ 0,66293564 -1,94013093 -0,78189238 1,44904122 -1,81131482]
 [ 0,4441061 -0,18751726 -2,58252033 0,23076863 0,12182448]
 [-0,60061323 0,39855851 -0,55612255 2,0201934 0,70525187]
 [-1,82925367 1,32004437 0,03226202 -0,79073523 -0,20750692]
 [-0,25756077 -1,37543232 -0,71369897 -0,13556156 -0,34918718]
 [ 0,26048374 2,49871398 1,01139237 -1,73242425 -0,67235417]
 [ 0,30351062 -0,45425039 -0,84046541 -0,60435352 -0,06281934]
 [ 0,43562048 0,66297676 1,76386981 -1,11794675 2,2012095 ]
 [-1.11051533 0,3462945 0,19136933 0,19717914 -1,78323674]
 [ 1.1219638 -0,04282422 -0,0142484 -0,73210071 -0,58364205]
 [-1,24046375 0,23368434 0,62323707 -1,66265946 -0,87481714]
 [ 0,19484897 0,12629217 -1,01575241 -0,47028007 -0,58278292]
 [ 0,16703418 -0,50993283 -0,90036661 2,33584006 0,96395524]
 [-0,72714199 0,39000914 -1,3215123 0,92744032 -1,44239943]
 [-2.30234278 -0,52677889 -0,09759073 -0,63982215 -0,51416013]
 [ 1,25338899 -0,58950956 -0,86009159 -0,7752274 2,24655146]
 [ 0,07553743 -1,2292084 0,46184872 -0,56390328 0,15901276]
 [-0,52090565 -2,42754589 -0,78354152 -0,44405857 1,16228247]
 [-1,21805132 -0,40358444 -0,65942185 0,76753095 -0,19664978]
 [-1,5866041 1,17100962 -1,50840821 -0,61750557 1,56003127]
 [ 1,33045269 -0,85811272 1,88869376 0,79491455 -0,96199293]
 [-2,34456987 0,1005953 -0,99376025 -0,94402235 -0,3078695 ]
 [ 0,93611909 0,58522915 -0,15553566 -1,03352997 -2,7210093 ]]

 [[-0,50650286 -0,41168428 -0,7107231 ]
 [ 1,86861492 -0,36446849 0,97721539]
 [-0,12792125 0,69578056 -0,6639736 ]
 [ 0,58190462 -0,98941614 0,40932723]
 [ 0,89758789 -0,49250365 -0,05023684]]
```

Figura 7 – Geração de pesos iniciais (a) e seus valores (b)

7. Treinando o Modelo

Agora que a estrutura está definida, as funções e os pesos inicializados, pode-se treinar o modelo usando a função *train*. Esta função atualizará os pesos por meio de retropropagação por um determinado número de épocas (*epochs*) [1]. A Figura 8 apresenta a sequência de código.

```
def train(x, Y, w1, w2, alpha = 0.01, epoch = 10):
    acc =[]
    losss =[]
    for j in range(epoch):
        l =[]
        for i in range(len(x)):
            out = f_forward(x[i], w1, w2)
            l.append((loss(out, Y[i])))
            w1, w2 = back_prop(x[i], y[i], w1, w2, alpha)
        print("epochs:", j + 1, "======== acc:", (1-(sum(l)/len(x)))*100)
        acc.append((1-(sum(l)/len(x)))*100)
        losss.append(sum(l)/len(x))
    return(acc, losss, w1, w2)

acc, losss, w1, w2 = train(x, y, w1, w2, 0.1, 100)
```

Figura 8 – Código para treinamento por retropropagação.

8. Precisão e perda de plotagem

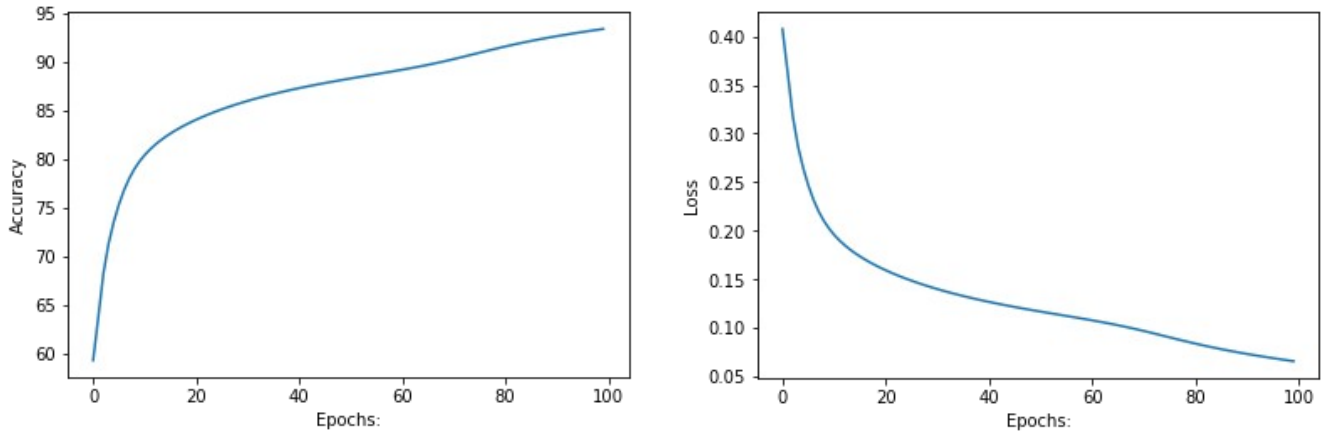
Após o treino, pode-se visualizar o precisão (*accuracy*) e perda (*loss*) ao longo das épocas para compreender o processo de aprendizagem do modelo, conforme Figura 9.

```
import matplotlib.pyplot as plt1

# plotting accuracy
plt1.plot(acc)
plt1.ylabel('Accuracy')
plt1.xlabel("Epochs:")
plt1.show()

# plotting Loss
plt1.plot(losss)
plt1.ylabel('Loss')
plt1.xlabel("Epochs:")
plt1.show()
```

(a)



(b)

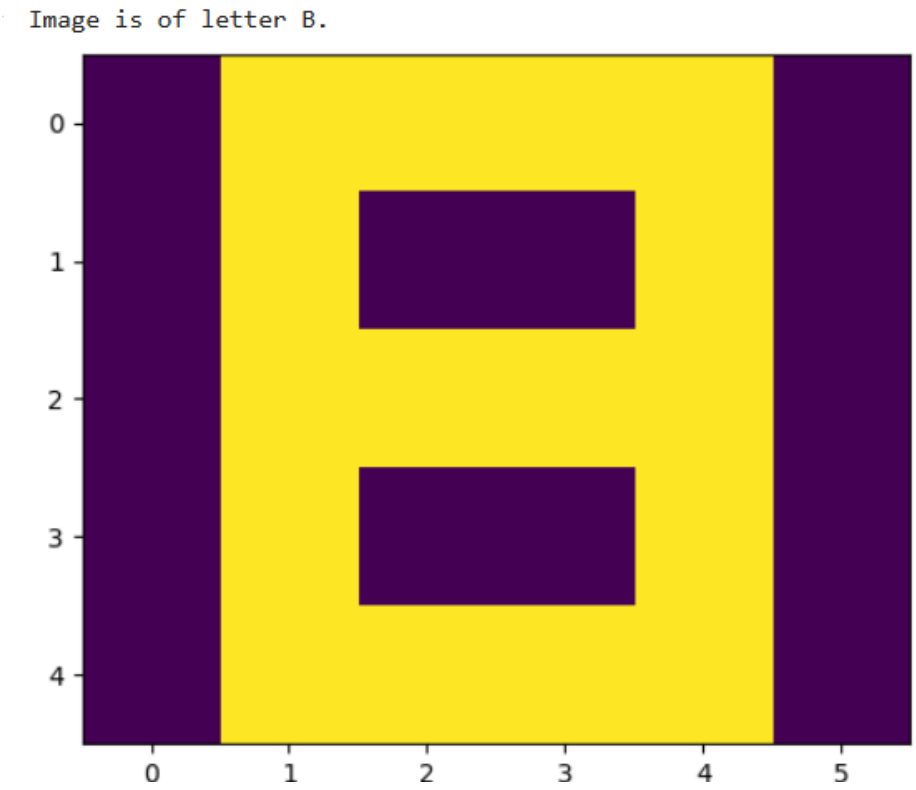
Figura 9 – Código para plotagem da precisão, perdas (a) saídas (b).

9. Fazendo previsões

Usam-se os pesos treinados para prever a classe de letras para uma nova entrada. A classe com maior valor de saída é escolhida como classe prevista, conforme a Figura 10 [1]..

```
def predict(x, w1, w2):
    Out = f_forward(x, w1, w2)
    maxm = 0
    k = 0
    for i in range(len(Out[0])):
        if(maxm<Out[0][i]):
            maxm = Out[0][i]
            k = i
    if(k == 0):
        print("Image is of letter A.")
    elif(k == 1):
        print("Image is of letter B.")
    else:
        print("Image is of letter C.")
    plt.imshow(x.reshape(5, 6))
    plt.show()
# Example: Predicting for letter 'B'
predict(x[1], w1, w2)
```

(a)



(b)

Figura 10 – Fazendo previsões (a) e plotando saída (b).

CONCLUSÃO

Este tutorial apresentou um algoritmo em Python de ANN para identificação e classificação de letras A, B e C, usando bibliotecas NumPy e Matplotlib. O código completo pode ser visto no ANEXO I. O link para o código em Google Colab está disponível em: clicando aqui.

REFERÊNCIAS

[1] GEEKSFORGEEKS, Implementation of Neural Netwok from Scratch Using NumPy, GeeksForGeks, 2025. Disponível em: <https://www.geeksforgeeks.org/implementation-of-neural-network-from-scratch-using-numpy/>. Acesso em Maio 20, 2025.

[2] DE CARVALHO JUNIOR, A. Função de Ativação, o Núcleo da Composição de Neurônios Artificiais, EAILAB, IFSP, 2024. Disponível em: <https://eailab.labmax.org/2024/02/28/funcao-de-ativacao-o-nucleo-da-composicao-de-neuronios-artificiais/>. Acesso em Maio 20, 2025.

[3] DE CARVALHO JUNIOR, A. Redes Neurais Artificiais: Algoritmos poderosos para aplicações de IA e ML, EAILAB, IFSP, 2024. Disponível em: <https://eailab.labmax.org/2024/04/03/redes-neurais-artificiais-algoritmos-poderosos-para-aplicacoes-de-ia-e-ml/>. Acesso em Maio 20, 2025.

[4] DE CARVALHO JUNIOR, A. Métricas de Desempenho em Modelos de IA (parte 1), EAILAB, IFSP, 2025. Disponível em: <https://eailab.labmax.org/2025/05/06/metricas-de-desempenho-em-modelos-de-ia-parte-1/>. Acesso em Maio 20, 2025.

[5] DE CARVALHO JUNIOR, A. Métricas de Desempenho em Modelos de IA (parte 2), EAILAB, IFSP, 2025. Disponível em: <https://eailab.labmax.org/2025/05/13/metricas-de-desempenho-em-modelos-de-ia-parte-2/>. Acesso em Maio 20, 2025.

ANEXO I

```
# Implementação de Rede Neural do Zero Usando NumPy
# Por: Geeks for Geeks
# Ref: https://www.geeksforgeeks.org/implementation-of-neural-network-from-scratch-using-numpy/

# Identificação e Classificação de A, B e C
# Adaptado por: Dr. Arnaldo de Carvalho Junior - Janeiro 2026

# Immportando Bibliotecas
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt1

# 1. iando o Dataset Usando Matrizes de 0s e 1s NumPy
# Criando o Dataset

# A
a =[0, 0, 1, 1, 0, 0,
  0, 1, 0, 0, 1, 0,
  1, 1, 1, 1, 1, 1,
  1, 0, 0, 0, 0, 1,
  1, 0, 0, 0, 0, 1]

# B
b =[0, 1, 1, 1, 1, 0,
  0, 1, 0, 0, 1, 0,
  0, 1, 1, 1, 1, 0,
  0, 1, 0, 0, 1, 0,
  0, 1, 1, 1, 1, 0]

# C
c =[0, 1, 1, 1, 1, 0,
  0, 1, 0, 0, 0, 0,
  0, 1, 0, 0, 0, 0,
  0, 1, 0, 0, 0, 0,
  0, 1, 1, 1, 1, 0]

# Criando os Rótulos
y =[[1, 0, 0],
  [0, 1, 0],
  [0, 0, 1]]

# 2. Visualizando o Dataset com Matplotlib
```

```
# visualizando os dados, plotando A.
plt.imshow(np.array(a).reshape(5, 6))
plt.show()

# 3. Como o dataset está na forma de lista, o converte-se em matriz NumPy.
# convertendo dados e rótulos em matriz NumPy
x =[np.array(a).reshape(1, 30), np.array(b).reshape(1, 30), np.array(c).reshape(1,
30)]
y = np.array(y)

# Imprimindo Dados e Rótulos (labels)
print(x, "\n\n", y)

# 4. Definindo a Arquitetura da Rede Neural Artificial
# input layer com 30 nós (5x6 grid)
# hidden layer com 5 nós
# output layer com 3 nós (representando as letras A, B e C)
# feedforward, backpropagation, MSE com função perda (loss)

# função de ativação (sigmoide)
def sigmoid(x):
 return(1/(1 + np.exp(-x)))

# 5. Criando a rede neural feed-forward
def f_forward(x, w1, w2):

 # camada oculta
 z1 = x.dot(w1)    # entrada da camada 1
 a1 = sigmoid(z1)  # saída da camada 2
 z2 = a1.dot(w2)   # entrada da camada de saída
 a2 = sigmoid(z2)  # saída da rede neural
 return(a2)

# inicializando os pesos aleatoriamente
def generate_wt(x, y):
 li =[]
 for i in range(x * y):
  li.append(np.random.randn())
 return(np.array(li).reshape(x, y))

# para perda será usado o erro médio quadrático (mean square error - MSE)
def loss(out, Y):
 s =(np.square(out-Y))
 s = np.sum(s)/len(y)
 return(s)
```

```
# retropropagação (backpropagation) do erro
def back_prop(x, y, w1, w2, alpha):

 # camada oculta (hidden layer)
 z1 = x.dot(w1)
 a1 = sigmoid(z1)
 z2 = a1.dot(w2)
 a2 = sigmoid(z2)

 # erro na camada de saída
 d2 =(a2-y)
 d1 = np.multiply((w2.dot((d2.transpose()))).transpose(),(np.multiply(a1, 1-a1)))

 # Gradiente para w1 e w2
 w1_adj = x.transpose().dot(d1)
 w2_adj = a1.transpose().dot(d2)

 # atualizando parâmetros
 w1 = w1-(alpha*(w1_adj))
 w2 = w2-(alpha*(w2_adj))

 return(w1, w2)
# 6. Initializing Weights
w1 = generate_wt(30, 5)
w2 = generate_wt(5, 3)

print(w1, "\n\n", w2)

# 7. Treinando o modelo
def train(x, Y, w1, w2, alpha = 0.01, epoch = 10):
 acc =[]
 losss =[]
 for j in range(epoch):
  l =[]
  for i in range(len(x)):
   out = f_forward(x[i], w1, w2)
   l.append((loss(out, Y[i])))
   w1, w2 = back_prop(x[i], y[i], w1, w2, alpha)
  print("epocas:", j + 1, "======== acc:", (1-(sum(l)/len(x)))*100)
  acc.append((1-(sum(l)/len(x)))*100)
  losss.append(sum(l)/len(x))
 return(acc, losss, w1, w2)

acc, losss, w1, w2 = train(x, y, w1, w2, 0.1, 100)
```

```
# 8. Plotando a Acurácia e Perda
# plotando acurácia
plt1.plot(acc)
plt1.ylabel('Acurácia:')
plt1.xlabel("Épocas:")
plt1.show()

# plotando perda
plt1.plot(losss)
plt1.ylabel('Perda')
plt1.xlabel("Épocas:")
plt1.show()

# 9. Realizando Predições
def predict(x, w1, w2):
 Out = f_forward(x, w1, w2)
 maxm = 0
 k = 0
 for i in range(len(Out[0])):
  if(maxm<Out[0][i]):
   maxm = Out[0][i]
   k = i
 if(k == 0):
  print("A imagem é a letra A.")
 elif(k == 1):
   print("A imagem é a letra B.")
 else:
  print("A imagem é a letra C.")
 plt.imshow(x.reshape(5, 6))
 plt.show()

# Exemplo: predizendo a letra "B"
predict(x[1], w1, w2)
```