Capítulo III - Desenvolvimento de um Neurônio Artificial: Um estudo de acadêmicos para acadêmicos.

Danilo Faveri Massi[8]
Diego André Gil [9]
Luis Gustavo Ferrareto Espontão [10]
Prof. Esp. Wesley Candido Silva [11]

RESUMO

Este artigo tem como objetivo fornecer uma introdução para que, a comunidade acadêmica dos cursos de Engenharia, possa praticar de forma perspicaz e criativa a análise de novas tecnologias e as oportunidades trazidas por essas; habilitando, a mesma, em pesquisa e desenvolvimento de conceitos, tais como: a construção de uma Rede Neural Artificial, abordar uma base de dados para aprendizagem do neurônio, identificar o modelo de neurônio, codificar o treinamento do neurônio artificial, utilizar métodos de ajuste de peso da função e obter os parâmetros de forma dinâmica; demonstrando, desta forma, a capacidade de desenvolvimento de uma Rede Neural Artificial de uma camada, programada em Python. Por fim, que possa aplicar estes conhecimentos matemáticos, científicos, tecnológicos e instrumentais em engenharia, considerando os aspectos humanísticos, sociais, éticos, legais, ambientais e econômicos fazendo uso do método científico, preconizados nos projetos pedagógicos da Faculdade da Indústria SENAI Londrina.

Palavras-chave: RNA. Neuronio. Implementação *Python*. Aprendizagem.

[8] Discente do programa de graduação em Engenharia Elétrica na Fac. Ind. SENAI Londrina.
[9] Discente do programa de graduação em Engenharia Elétrica na Fac. Ind. SENAI Londrina.
[10] Discente do programa de graduação em Engenharia Elétrica na Fac. Ind. SENAI Londrina.
[11] Docente do programa de pós-graduação e graduação na Fac. Ind. SENAI Londrina. E-mail: Wesley.candido@sistemafiep.org.br

Development of an Artificial Neuron: a study by academics for academics

ABSTRACT

This article aims to provide an introduction so that the academic community of Engineering courses can practice insightfully and creatively the analysis of new technologies and the opportunities brought by these; enabling the same in research and development of concepts, such as: the construction of an Artificial Neural Network, approaching a database for neuron learning, identifying the neuron model, coding the training of the artificial neuron, using adjustment methods function weight and get the parameters dynamically; demonstrating, in this way, the ability to develop a one-layer Artificial Neural Network, programmed in Python. Finally, that you can apply this mathematical, scientific, technological and instrumental knowledge in engineering, considering the humanistic, social, ethical, legal, environmental and economic aspects, making use of the scientific method, recommended in the pedagogical projects of the Faculty of Industry SENAI Londrina.

Keywords: RNA. Neuron. Python implementation. Learning.

1. INTRODUÇÃO

As redes neurais artificiais são antigas, e tem se algumas informações históricas que datam de 1943, segundo Furtado (2019), as primeiras informações sobre a neuro computação datam de 1943, em artigos de McCulloch e Pitts, em que sugeriam a construção de uma máquina baseada ou inspirada no cérebro humano. – A partir dessa data foram surgindo livros, construídos alguns computadores neuros, além de estudos e pouco desenvolvimento, até que em 1983, foi fundado um centro de pesquisa em neuro computação, e em 1987 muitas universidades anunciaram formações de institutos, para pesquisa e desenvolvimento.

As redes neurais artificiais, (RNAs) utilizam uma espécie de neurônio artificiais programáveis, que fazem quase que a mesma função de um neurônio humano, porém, foi baseado no humano. Segundo Moreira (2017), os neurônios são unidades do sistema nervoso, que recebem informações (sinais elétricos) de outros neurônios e de neuro receptores especializados, integrando estas informações em suas áreas

operacionais e encaminhando-as, ao final do processo, na forma de uma mensagem, em direção a outros neurônios ou para estruturas efetoras, músculos ou glândulas. – São desta forma que funcionam os neurônios humanos, já o neurônio artificial segundo Furtado (2019), é composta por um elevado número de elementos processadores, os neurônios, amplamente interligados através de conexões com um determinado valor que estabelece o grau de conectividade.

Segundo Carneiro (2001), existem diversas aplicações para redes neurais artificias, tanto que o volume de investimentos nesta área tem alcançado somas expressivas tanto no Brasil quanto no exterior. Pode-se destacar as aplicações nas áreas médicas, financeiras, robótica, aeroespacial etc. – Sendo assim, temos uma infinidade de aplicações que podem ser utilizadas, e melhorias feitas para melhor desenvolvimento da sociedade das tecnologias.

2. METODOLOGIA

Para a implementação de um neurônio artificial de apenas uma camada, foi realizado utilizando Python através do *jupyter* notebook, trata-se de um ambiente de desenvolvimento *opensource,* amplamente utilizado por estudantes de programação e ciência de dados.

O primeiro passo para realizar a implementação de um neurônio artificial, se trata de ter uma base de dados para realizar o treinamento, que possibilita o neurônio realizar predições através do aprendizado obtido pela base dos dados. Para esta implementação será utilizado uma base de dados para identificação de cores para pintura de peças de determinados tamanhos, como segue a Tabela 1.

Tabela 1 – Base de dados

Largura	Comprimento	Cor
3	1.5	Vermelho
2	1	Azul
4	1.5	Vermelho
3	1	Azul
3.5	0.5	Vermelho
4	0.5	Vermelho

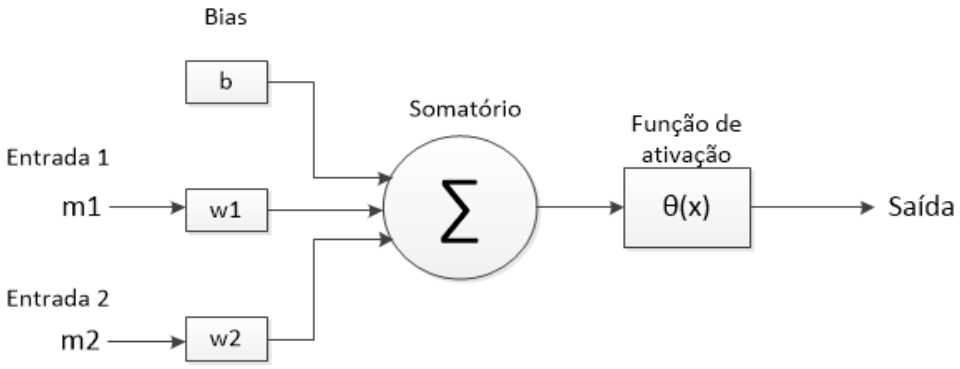
5.5	1	Vermelho
1	1	Azul

Fonte: Dos Autores.

Para codificarmos a Tabela 1 de forma que seja possível para o computador processar, devemos converter as saídas (Cor) para uma forma Booleana, visto que temos apenas duas opções, no caso foi definido que 1 indica vermelho e 0 indica azul.

Com a base de dados obtida e normalizada para uma linguagem que seja possível uma rede neural receber e processar os dados, o próximo passo é identificar o modelo de neurônio ideal para a aplicação, que pode ser verificado que o neurônio deverá ter no mínimo duas entradas e uma saída para retornar o resultado seguindo a base de dados, neste caso também foi inserido uma entrada extra para uma variável “Bias” com a estratégia de auxiliar ao modelo a se adaptar melhor aos dados, e apresenta uma saída que não seja nula (HAYKIN, 2001).

Figura 1 – Modelo Neurônio Artificial



Fonte: Dos Autores.

Como ilustrado pela Figura 1, o modelo proposto é dado por 3 variáveis de entrada, sendo elas, respectivamente: Bias, m1 e m2. Após as variáveis serem balanceadas pelos pesos w1 e w2, os valores passam por uma somatória onde são enviados a uma função de ativação do neurônio, que retorna a saída dele.

Para o neurônio artificial tomar uma decisão ele necessita de uma função que ative ele ou seja uma função de ativação por exemplo função de limiar, função rampa, função sigmoide, entre outras.

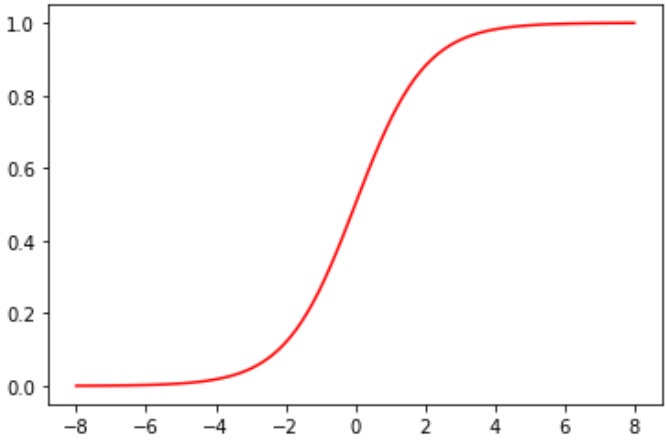
Segundo HAYKIN (2001), neurônios biológicos funcionam de forma binária (ativado ou não ativado), a função sigmoide (Figura 2) é uma boa forma de modelar

esse comportamento, já que assume valores apenas entre 0 (não ativação) e 1 (ativação). Para esta aplicação será utilizado a função sigmoide devido a simplicidade de implementação e fornecer uma resposta de propagação positiva, ou seja, a saída apenas para números positivos, expressa matematicamente pela equação 1.

$$Y(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

Sendo Y a reposta da função sigmoide a uma entrada x, sendo o valor resultante da somatória do neurônio.

Figura 2 – Função Sigmoide



Fonte: Dos Autores.

Será codificado o modelo desse neurônio utilizando a linguagem de programação Python, usando bibliotecas com funções matemáticas e para plotar os gráficos para a análise dos dados (Figura 3).

Figura 3 – Bibliotecas de função matemáticas e plotagem de gráficos

```
#Blibioteca para funções matematicas
%matplotlib inline
import numpy as np
from matplotlib import pyplot as plt
```

Fonte: Dos Autores.

Definição das funções de somatória e ativação do neurônio (Figura 4).

Figura 4 – Funções do Neurônio Artificial

```
def NN(m1,m2,w1,w2,b):
    z = m1 * w1 + m2 * w2 + b
    return z
#Função Sigmoid
def sigmoid(x):
    return 1/(1 + np.exp(-x))
```

Fonte: Dos Autores.

Sendo NN a função somatória, que tem como parâmetros ilustrados pela Figura 1, e retornando o valor da somatória, e a função sigmoide sendo expressa da mesma maneira que a equação 1 por meio do uso das bibliotecas ilustradas na Figura 3.

Desta forma nosso neurônio está codificado e já é possível realizar predições, inserindo os valores para cada parâmetro das funções e identificando o retorno da função sigmoide em relação ao valor da somatória do neurônio, entretanto, este valor de predição não será preciso, pois não realizamos o treinamento deste neurônio, ou seja, obter os valores ideais para cada peso “w1”,” w2” e “b” para obter a resposta desejada para nossa entrada de dados.

Para este treinamento existem métodos empíricos, onde é definido um “passo” para cada valor de peso, e vai alterando até se ajustar para uma saída satisfatória, porém para casos em que temos uma grande quantidade de dados, com muitos pesos envolvidos e deseja-se uma resposta precisa, é necessária uma rotina de treinamento dinâmico, onde busca-se os valores de peso para cada variável, por meio de métodos de cálculo diferencial, buscando a menor taxa de erro.

Nesta aplicação iremos utilizar funções de Custo (*Cost Function*), que resulta o quão assertivo nosso modelo está com as atuais circunstâncias, em outras palavras, é a distância que nossa predição está do valor alvo, expresso pela equação 2.

$$Erro = (Predição - Alvo)^2 \quad (2)$$

Por exemplo, digamos que neste caso nosso Alvo seja o valor 1, então nossa predição tem que se aproximar muito de 1, para que a diferença ao quadrado seja zero ou muito próxima a zero.

Agora vemos um desafio de como saber quando devemos aumentar ou diminuir nossos parâmetros para melhorar nossas saídas, dessa forma a maneira utilizada para este tipo de função, é basicamente calcular a inclinação da curva do erro, assim saberemos quando estamos aproximando do erro zero, ou próximo, aumentando ou diminuindo o valor de nossos pesos.

Para isso vamos utilizar métodos de cálculo, no caso sabemos que a derivada é a inclinação da reta tangente que passa por uma curva, onde no caso é exatamente o que desejamos saber, ao realizarmos a derivada na equação 2, obtemos a equação 3.

$$Inclinação\ = 2\ (Predição - Alvo) \tag{3}$$

Ao analisar o comportamento das funções 2 e 3, é possível notar que quando nosso alvo for 1, e maior for nossa predição, mais tangencial ao plano nossa curva irá estar, pois se aproximara de zero, logo saberemos que estamos no ponto ideal da nossa curva de erro e dos parâmetros de pesos.

Por fim, para atualizar dinamicamente nossos parâmetros, podemos aplicar um passo a função de atualização, ou seja, multiplicar o valor de nossa inclinação por uma constante (chamado de *learning rate*) e subtrair de nosso erro, assim a cada "passo" de nossa função irá tender a 0, ou seja, tangente ao plano x. Na aplicação foi realizado vários testes, ajustando o learning rate e a quantidade de iteração do loop até encontrar um resultado satisfatório.

A implementação deste treinamento em Python segue ilustrado pela Figura 5.

Figura 5 – Loop de Treinamento

```
#Loop de treinamento
learnig_rate = 0.2 # "passo" ou constante que multiplica o valor da inclinação
costs = []

#Assume valores randomicos
w1 = np.random.randn()
w2 = np.random.randn()
b = np.random.randn()

for i in range(50000): #Loop de 50000 iteração, ou seja a rotina de treino irá rodar em um laço de 50000 vezes.

    Sample = np.random.randint(len(data)) #Retorna um valor aleatório da matriz de dados
    Dados = data[Sample]
    z = NN(Dados[0],Dados[1],w1,w2,b)
    pred = sigmoid(z)

    target = Dados[2]

    error = cost(pred, target) #Calculo do erro

    inclin = slope(pred, target) #Inclinaração da reta do erro

    dpred_dz = d_sigmoid(z) #Calcula a derivada da predição

    dz_dw1 = Dados[0] #Derivada de w1
    dz_dw2 = Dados[1] #Derivada de w2
    dz_b = 1 #Derivada de b

    #Calcula os erros para cada parametro por meio da multiplicação da inclinação da curva do erro, a derivada da predição
    # e a derivada do parametro a ser calculado, assim obtem-se a derivada do erro em relação ao parametro desejado
    dcost_dw1 = inclin * dpred_dz * dz_dw1
    dcost_dw2 = inclin * dpred_dz * dz_dw2
    dcost_b = inclin * dpred_dz * dz_b

    #Calcula o novo peso para cada parametro, subtraindo o learning rate, e multiplicando pela parcela de erro do peso
    #calculado anteriormente
    w1 = w1 - learnig_rate * dcost_dw1
    w2 = w2 - learnig_rate * dcost_dw2
    b = b - learnig_rate * dcost_b

    #A cada 100 iterações do loop, coleta uma amostra, realiza uma predição, calcula o erro (cost) e armazena
    #no vetor e realiza a média de erros e plota no gráfico.
    if i % 100 == 0:
        const_sum = 0
        for j in range(len(data)):
            Dados = data[Sample]

            z = NN(Dados[0],Dados[1],w1,w2,b)
            pred = sigmoid(z)

            target = Dados[2]
            const_sum += cost(pred, target)

        costs.append(const_sum/len(data))
plt.plot(costs)
```

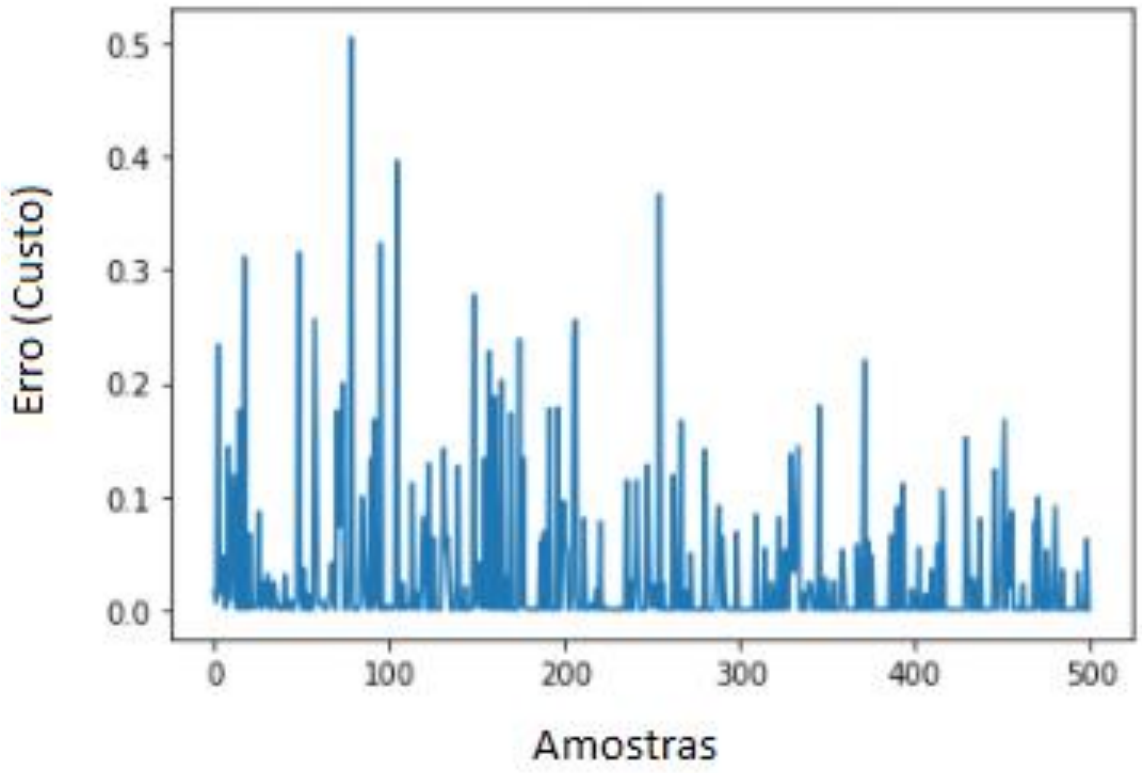
Fonte: Dos Autores.

Seguindo esta metodologia, resumidamente, foi realizada a implementação do neurônio artificial e foi realizado uma coleta dos dados de saída por meio da plotagem da curva de aprendizagem do neurônio e suas respectivas saídas para a base de dados para validação do treinamento.

3 CONCLUSAO

Os resultados obtidos com o treinamento do neurônio artficial foram satifatórios, como segue a ilustração da curva de aprendizado (Figura 6).

Figura 6 – Curva de aprendizagem



Fonte: Dos Autores.

Analisando o grafico vemos que a partir de 400 amostras, a curva de erro começou a diminuir para um nivel de aproximadamente 0,1 de erro, que para a aplicação foi totalmente satisfatória, por ser com o foco de auxiliar no aprendizado e implementação de algoritmos inteligentes. Para validar esse treinamento, também foi realizado teste com os dados do treianamento, e plotados sequencialmente com o dado analisado e a predição do neurônio (Figura 6).

Por se tratar de uma implementação de um sistema que tenta copiar a complexidade de um neurônio do ser humano, a base de conhecimento e estudo a cerca do tema pode ser muito desafiador, por conter muitas aplicações de conceitos de estaticas e calculo, como pode ser visto na metodologia deste trabalho.

Figura 6 – Resultados das predição do Neurônio Artificial

```
[3, 1.5, 1]
Pred: 0.7014570549857383
[2, 1, 0]
Pred: 0.000568134260863994
[4, 1.5, 1]
Pred: 0.9994705814285146
[3, 1, 0]
Pred: 0.31353833397575315
[3.5, 0.5, 1]
Pred: 0.715647787964623
[4, 0.5, 1]
Pred: 0.9861763234026596
[5.5, 1, 1]
Pred: 0.9999998803578608
[1, 1, 0]
Pred: 7.074909868003077e-07
```

Fonte: Dos autores.

O principal foco é a difusão do conceitos para os alunos e colegas de classe da graduação, ou qualquer estudante que esteja realizando pesquisas sobre o tema. Desta forma é nítido que este trabalho tem a contribuição apenas de realizar um pequeno mergulho no ramo de ciencia de dados e redes neurais, onde foi passado por vários conceitos de maneira resumida que é possivel encontrar estudos apenas voltados para o conceito aplicado.

Por fim, para auxilio do estudante interessado em implementar o neurônio artificial aplicado na metodologia deste trabalho, o código é de licença livre, encontrando-se no anexo A deste documento.

REFERÊNCIAS

HAYKIN, S. **Redes Neurais, princípios e práticas**. Porto Alegre: Bookman, 2001. Disponivel em: https://www.deeplearningbook.org.

Moreira, É. S. **Os neurônios, as sinapses, o impulso nervoso e os mecanismos morfo-funcionais de transmissão dos sinais neurais no sistema nervoso**. Volta Redonda: UniFOA, 2017. v.2. p.81 Il. Disponivel em http://editora.unifoa.edu.br/wp-content/uploads/2017/04/Volume-02.pdf.

Furtado, M. I. V. **Redes neurais artificiais**: uma abordagem para sala de aula. Ponta Grossa: Atena, 2019. Disponivel em: https://www.atenaeditora.com.br/wp-content/uploads/2019/05/e-book-Redes-Neurais-Artificiais-uma-Abordagem-para-Sala-de-Aula.pdf.

Carneiro, D. L. **Um estudo sobre a aplicabilidade de redes neurais em criptografia**. UFSC, Florianópolis 2001

ANEXO A

```
#Blibioteca para funções matematicas
%matplotlib inline
import numpy as np
from matplotlib import pyplot as plt


#Dados para treinamento - 1 = vermelho, 0 = azul
#Os dados se tratam de medidas de produtos de uma máquina que são de
tamanhos diferentes
#que precisam ser selecionados por cor para identificação.

data = [[3,   1.5, 1],
        [2,   1,   0],
        [4,   1.5, 1],
        [3,   1,   0],
        [3.5, 0.5, 1],
        [4,   0.5, 1],
        [5.5, 1,   1],
        [1,   1,   0]]


#Função que define nosso neurônio com 2 entradas e 2 pesos para entrada,
com a função de ativação Sigmoid, que retorna valores de 0 a 1.
def NN(m1,m2,w1,w2,b):
    z = m1 * w1 + m2 * w2 + b
    return z
#Função Sigmoid
def sigmoid(x):
    return 1/(1 + np.exp(-x))
#Derivada da função Sigmoid
def d_sigmoid(x):
    return sigmoid(x) * (1 - sigmoid(x))


#Plota os gráficos das funções Sigmoide e Derivada da Sigmoide
T = np.linspace(-8,8,100)
plt.plot(T,sigmoid(T), c='r')
#plt.plot(T,d_sigmoid(T), c='b')


#Função para obter o erro das predições
def cost(pred, target):
    return np.square(pred - target)


#Função para obter a inclinação da curva (derivada do erro (cost))
def slope(pred, target):
    return 2 * (pred - target)
```

```
#Loop de treinamento

learnig_rate = 0.2
costs = []

w1 = np.random.randn()
w2 = np.random.randn()
b = np.random.randn()

for i in range(50000):

    Sample = np.random.randint(len(data)) #Retorna um valor aleatório da
matriz de dados
    Dados = data[Sample]
    z = NN(Dados[0],Dados[1],w1,w2,b)
    pred = sigmoid(z)

    target = Dados[2]

    error = cost(pred, target)

    inclin = slope(pred, target)
    dpred_dz = d_sigmoid(z)

    dz_dw1 = Dados[0]
    dz_dw2 = Dados[1]
    dz_b = 1

    dcost_dw1 = inclin * dpred_dz * dz_dw1
    dcost_dw2 = inclin * dpred_dz * dz_dw2
    dcost_b = inclin * dpred_dz * dz_b

    w1 = w1 - learnig_rate * dcost_dw1
    w2 = w2 - learnig_rate * dcost_dw2
    b = b - learnig_rate * dcost_b

    #A cada 100 iterações do loop, coleta uma amostra, realiza uma
predição, calcula o erro (cost) e armazena
    #no vetor e realiza a média de erros e plota no gráfico.
    if i % 100 == 0:
        const_sum = 0
        for j in range(len(data)):
            Dados = data[Sample]

            z = NN(Dados[0],Dados[1],w1,w2,b)
            pred = sigmoid(z)

            target = Dados[2]
            const_sum += cost(pred, target)

        costs.append(const_sum/len(data))

plt.plot(costs)


#Valida os dados obtidos no treinamento do modelo
```

```
for i in range(len(data)):
    Dados = data[i]
    print(Dados)
    z = NN(Dados[0],Dados[1],w1,w2,b)
    pred = sigmoid(z)
    print("Pred: {}".format(pred))
```