

Criando um Perceptron do zero com Python



Alef Matias · [Follow](#)

7 min read · Jul 11, 2023



Share

Introdução

Seja você um indivíduo ligado ou não no mundo da tecnologia, é quase certo que em algum momento nesse mundo conectado você já se esbarrou com alguns termos tecnológicos como Inteligência Artificial, Redes Neurais, Aprendizado de Máquina (ou *Machine Learning* para os mais íntimos) e afins. Mas em algum momento você já se perguntou como essas coisas funcionam? Para os casos afirmativos, aqui estou eu recém formado em Ciência de Dados e pronto para me aprofundar mais em temas que a faculdade pecou durante os árduos 3 anos de curso.

Partindo desta breve introdução, vamos ao que de fato interessa: como criar um modelo de Perceptron do zero. A ideia deste artigo é explicar o que é este modelo e como suas entranhas funcionam passo a passo. Mas antes de tudo, o que é o Perceptron?

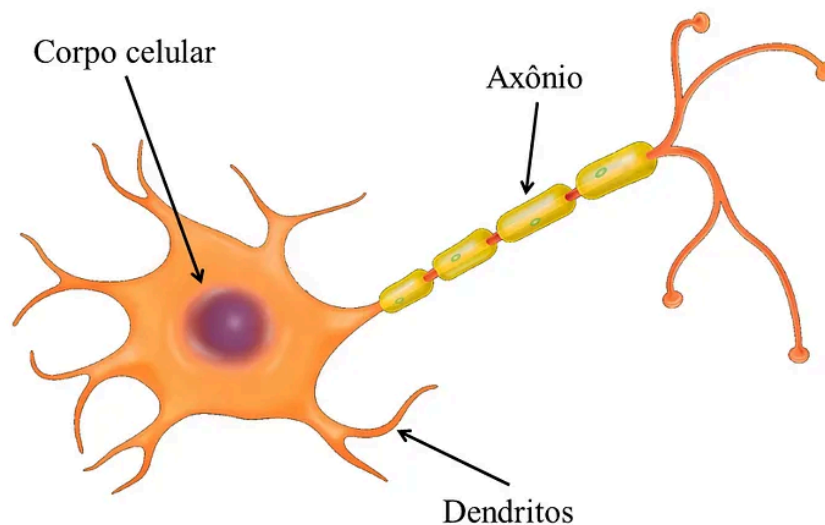
História e características

Criado em 1958 por Frank Rosenblatt, o Perceptron foi o primeiro modelo computacional de aprendizado de máquina, ou melhor dizendo, o modelo mais simples ou rascunho do que viria se tornar uma Rede Neural Artificial (RNA) como é

conhecida nos dias atuais. Sua criação servia para ser utilizada como um classificador linear ou classificador binário, podendo separar os dados em duas classes distintas, Sim e Não, Verdadeiro e Falso, 0 e 1, etc.

A ideia de Rede Neural Artificial foi inspirada na biologia, fazendo associação ao neurônio biológico. O neurônio é composto dos seguintes componentes:

- Dendritos: atua como receptor dos estímulos e leva o impulso nervoso ao corpo celular.
- Corpo celular: local onde está o núcleo do neurônio e grande parte de suas organelas.
- Axônio: prolongamento do neurônio responsável por conduzir o impulso nervoso.



Representação de um neurônio biológico e suas principais estruturas.

De forma simplificada, podemos retratar o neurônio biológico da seguinte forma: recebimento dos dados de entrada → processamento dos estímulos pelo núcleo → envio dos dados pela camada de saída.

Modelo de dados

Antes de iniciar o tutorial, acho interessante ressaltar que o modelo aqui apresentado foi pensado para processar as portas lógicas OR e AND com apenas duas dimensões, pois são o exemplo mais simples para testarmos o modelo, então o código não está otimizado para mais dimensões de dados. Como a ideia é tornar o entendimento o mais simples possível, achei interessante também utilizar dados simples para serem processados.

Ok, mas o que são portas lógicas? Eu te explico no bloco abaixo.

Portas lógicas são circuitos que operam com sinais lógicos de entrada e dependendo do seu tipo produzem uma saída específica. As portas lógicas estão presentes no estudo de lógica matemática, lógica *booleana* e talvez a mais conhecida de todas, a tabela verdade. Para facilitar o entendimento, vamos aos exemplos das portas OR e AND.

OR

Recebendo os dois sinais de entrada (0 ou 1 em cada entrada), a saída será verdadeira se uma ou mais entradas forem verdadeiras.

A	B	Saída
1	1	1
1	0	1
0	1	1
0	0	0

Tabela verdade com operador OR.

AND

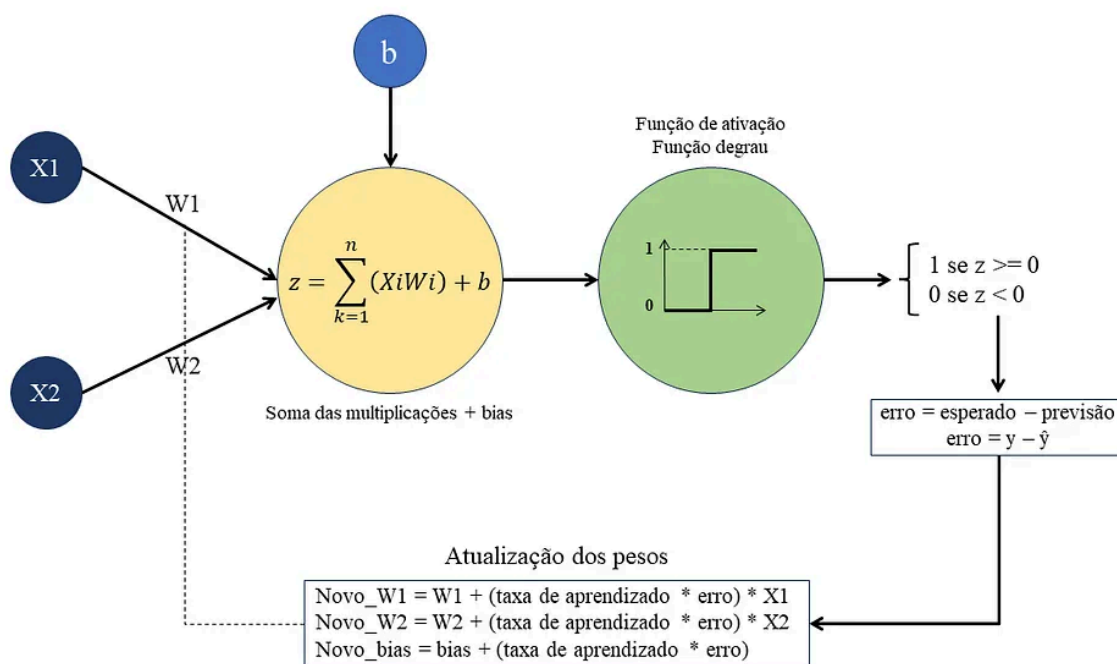
Recebendo os dois sinais de entrada (0 ou 1 em cada entrada), a saída será verdadeira apenas se as duas entradas forem verdadeiras.

A	B	Saída
1	1	1
1	0	0
0	1	0
0	0	0

Tabela verdade com operador AND.

Criando o modelo com Python

Passado os blocos com alguns conceitos básicos essenciais para o entendimento do algoritmo, vamos à criação do Perceptron. Assemelhando-se ao neurônio biológico, o modelo proposto por Frank Rosenblatt conta com a estrutura de entrada de dados, a estrutura de processamento e a estrutura de saída.



Arquitetura do Perceptron.

Onde:

- **x1 e x2:** dados de entrada.
- **w1 e w2:** pesos que serão multiplicados pelos dados de entrada.
- **b:** bias, também é um peso, porém é chamado de “peso livre” por não ser multiplicado com nenhuma outra variável.
- **z:** soma das multiplicações e adição do bias e posteriormente aplicado a função de ativação, que é responsável pela saída do dado processado.
- **função de ativação:** função matemática responsável por produzir o dado binário de saída. Normalmente na função de ativação é utilizada a função degrau, onde se a saída da função for maior ou igual a 0, o resultado produzido será 1, se for menor, resultará em 0.
- **erro:** valor esperado subtraído do valor predito pelo modelo.
- **taxa de aprendizagem:** taxa em que os pesos serão atualizados em cada iteração, valor entre 0 e 1.

Iniciamos o código fazendo a importação da biblioteca numpy, ela será utilizada apenas para criação dos pesos aleatórios iniciais entre -1 e 1 e realizar o cálculo exponencial. Logo após fazemos a inicialização da classe Perceptron.

```
import numpy as np

class Perceptron:
    # Aqui definimos o construtor da classe
    def __init__(self):
        pass
```

Definimos a função de treinamento do algoritmo, inicializando as variáveis necessárias.

```
def train(self, inputs, outputs, learning_rate, epochs):
    self.inputs = inputs
    self.outputs = outputs
    self.learning_rate = learning_rate
```

```
self.epochs = epochs

# Os pesos iniciais devem ser aleatórios na primeira rodada de treinamento
w1, w2, bias = np.random.uniform(-1, 1), np.random.uniform(-1, 1), np.random.
```

O laço de repetição abaixo será executado para de fato treinar o algoritmo durante a quantidade de épocas definida na função. O segundo laço tem a função de passar pelos dados de entrada, executar a função de ativação e ir atualizando os pesos até , que o resultado previsto seja o mais próximo possível do esperado.

```
for i in range(epochs):
    for j in range(len(inputs)):
        # Aqui definimos a função de ativação, no exemplo citado no início do artigo
        sigmoid = 1 / (1 + np.exp(- ((inputs[j][0] * w1) + (inputs[j][1] * w2) + bias)))

        # Atualizando os pesos após cada iteração
        w1 = w1 + (learning_rate * (outputs[j][0] - sigmoid) * inputs[j][0])
        w2 = w2 + (learning_rate * (outputs[j][0] - sigmoid) * inputs[j][1])
        bias = bias + (learning_rate * (outputs[j][0] - sigmoid))
    return w1, w2, bias
```

Importante ressaltar que após o treinamento ser executado, a função irá devolver os últimos pesos calculados após todas as iterações, é a partir deles que iremos fazer as previsões na próxima função que é a de predição.

```
def predict(self, weights, x1, x2):
    # Na função degrau devolvemos 1 se o resultado for maior ou igual a 0 e 0 se for menor
    # Na função sigmoid devolvemos 1 se o resultado for maior que 0.5 e 0 se for menor
    return 1 if 1 / (1 + np.exp(- ((x1 * weights[0]) + (x2 * weights[1]) + weights[2]))) > 0.5 else 0
```

Finalizando a construção da classe com suas funções de treinamento e predição, já podemos testar o algoritmo com dados que representam a tabela verdade com os operadores OR e AND.

```
# Cada item da lista dos dados de entrada representa as duas entradas lógicas
# A variável outputs representa a saída esperada da operação lógica, aqui no caso
inputs = [[0, 0], [0, 1], [1, 0], [1, 1]]
outputs = [[0], [1], [1], [1]]

percp = Perceptron()
# Passamos os dados de entrada e de saída, junto com a taxa de aprendizagem e a
train = percp.train(inputs, outputs, 0.01, 10000)
# Como a função train retorna os 3 pesos, passamos a variável train no método predict
predc = percp.predict(train, 1, 0)
predc
```

Considerações finais

De repente a seguinte dúvida poderá surgir: já que estamos passando os dados de entrada e suas respectivas respostas, por que então fazer toda essa volta para verificar um resultado que já sabemos e que ainda assim tem uma probabilidade (ainda que baixa nessa demonstração, dependendo da quantidade de épocas em que se foi configurada) de erro? E é aí que entra o aprendizado de máquina!

Se passamos somente o resultado que sabemos, a máquina não terá aprendido nada, apenas exibirá uma resposta previamente configurada, ela estará apenas consultando um dado. No aprendizado de máquina os pesos são recalculados

Open in app ↗

Sign up

Sign in



Search



aprender fazendo esses ajustes época por época.

Aqui demonstrei o exemplo mais simples, sendo até possível realizarmos os cálculos manualmente e ver a mágica acontecer, mas e quando tratamos de um número maior de variáveis que queremos mensurar e com um extenso número de dados? Aí a coisa começa a complicar um pouco mais. É nesse momento que nosso amigo *machine learning* entra em ação e pode nos auxiliar de forma mais eficiente.

Espero que tenha conseguido transmitir um pouco de conhecimento sobre o assunto através deste breve artigo, o primeiro aliás.

Até mais!

Machine Learning

Perceptron

Data Science

Neural Networks



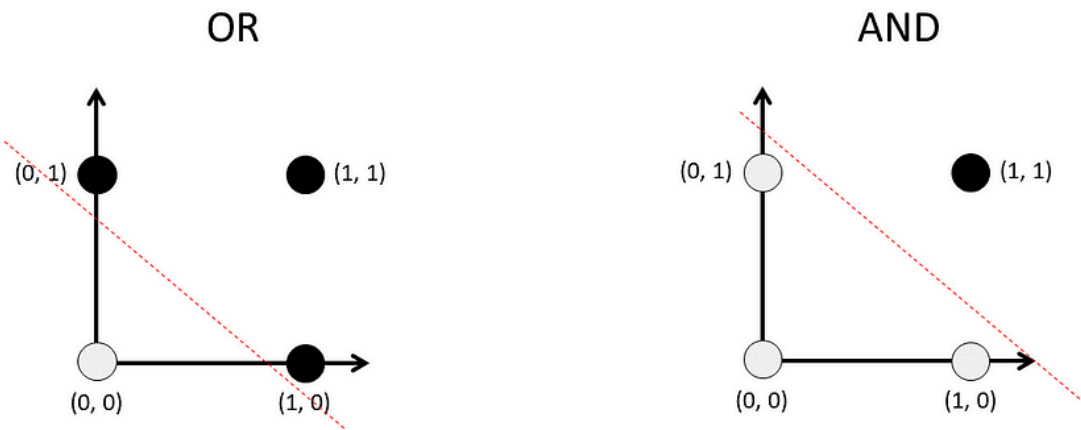
Follow

Written by Alef Matias

3 Followers

Cientista de Dados e Pós Graduando em Inteligência Artificial.

More from Alef Matias

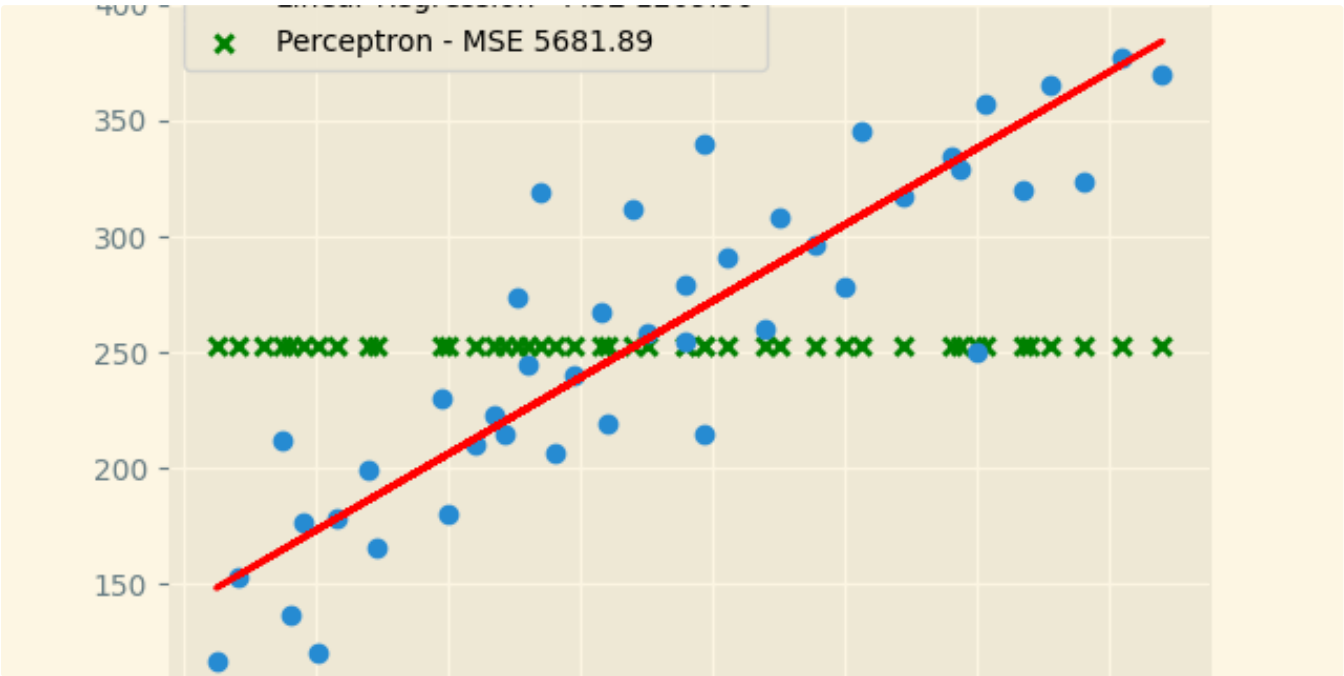


Alef Matias

Perceptron multicamadas do zero em Python—Porta lógica XOR

Considerações iniciais

6 min read · Jul 24, 2023



 Alef Matias

Comparação entre Regressão Linear e Rede Neural (Perceptron)

Conceitos iniciais

5 min read · Jan 21, 2024





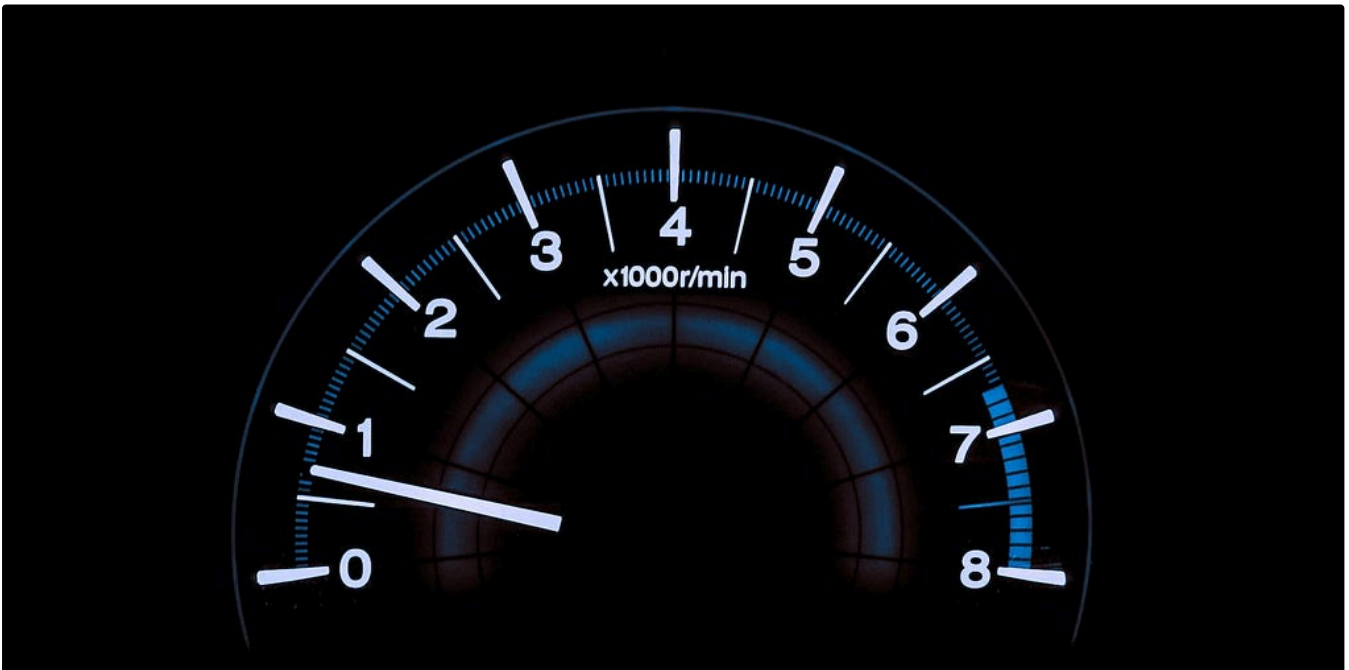
 Alef Matias

Como um CPF é validado?—Algoritmo em Python

Introdução

4 min read · Oct 23, 2023

 6 



 Alef Matias

Comparativo de velocidade entre linguagens de programação—Python vs. C

Blá blá blá inicial

3 min read · Sep 13, 2023

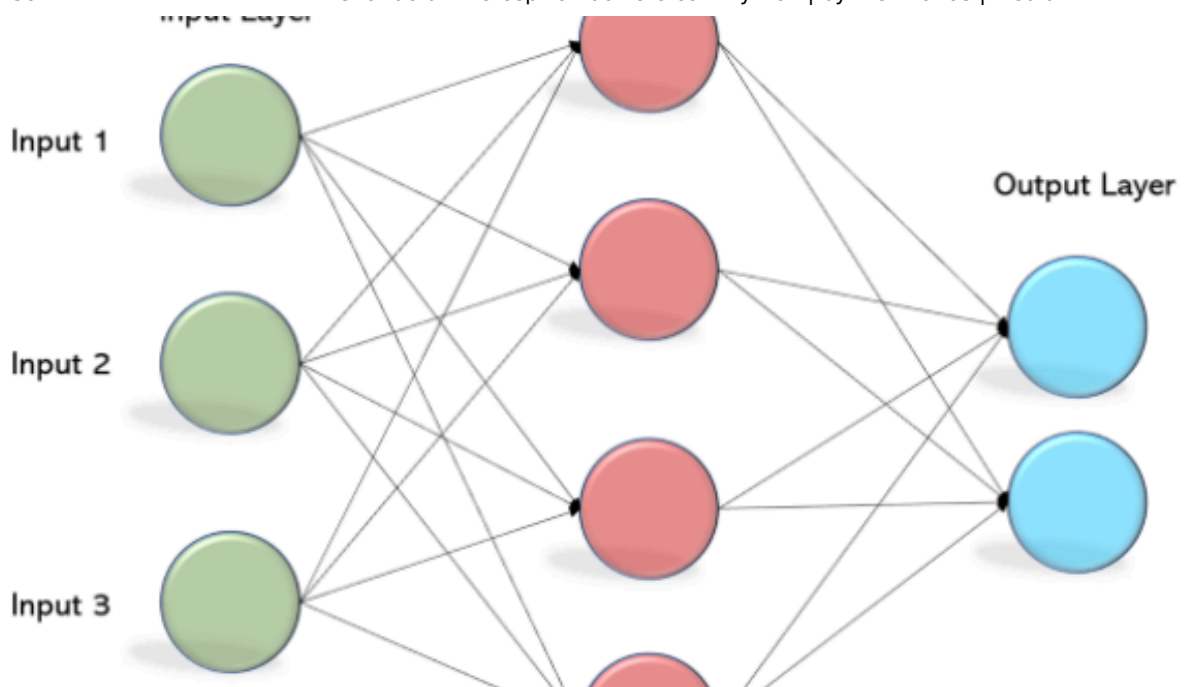
 1





See all from Alef Matias

Recommended from Medium



 Marcus Sena in Python in Plain English

A Step-by-Step Guide to Implementing Multi-Layer Perceptrons in Python

In the previous post, I showed how to build a single perceptron with python. Also, it was presented its limitations. This post aims to show...

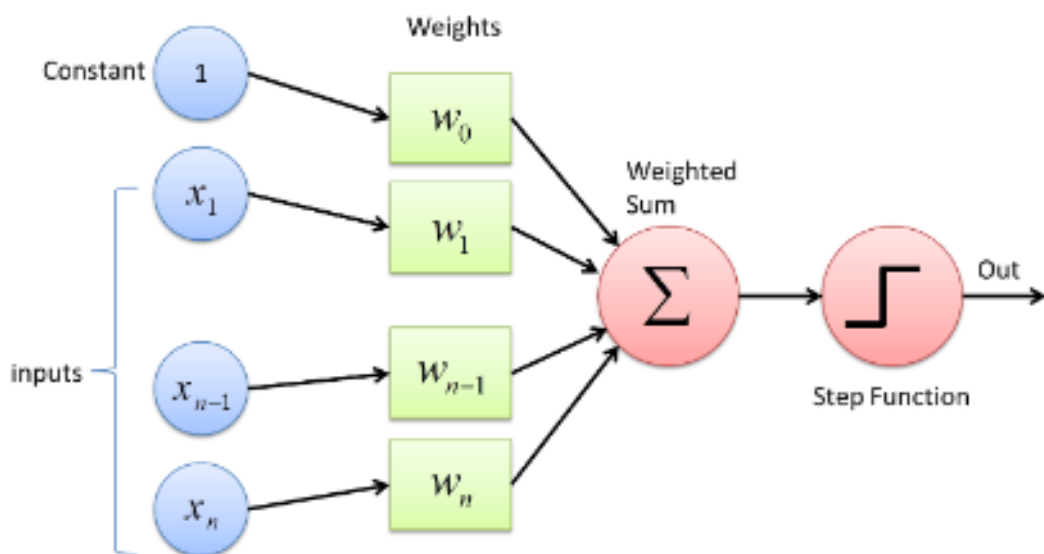
7 min read · Feb 10, 2024




15



1



 Joseph Robinson, Ph.D. in Towards Data Science

From Basic Gates to Deep Neural Networks: The Definitive Perceptron Tutorial

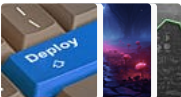
Demystifying Mathematics, Binary Classification, and Logic Gates

🌟 · 21 min read · Apr 28, 2023

👏 466 💬 3



Lists



Predictive Modeling w/ Python

20 stories · 1035 saves



Practical Guides to Machine Learning

10 stories · 1237 saves



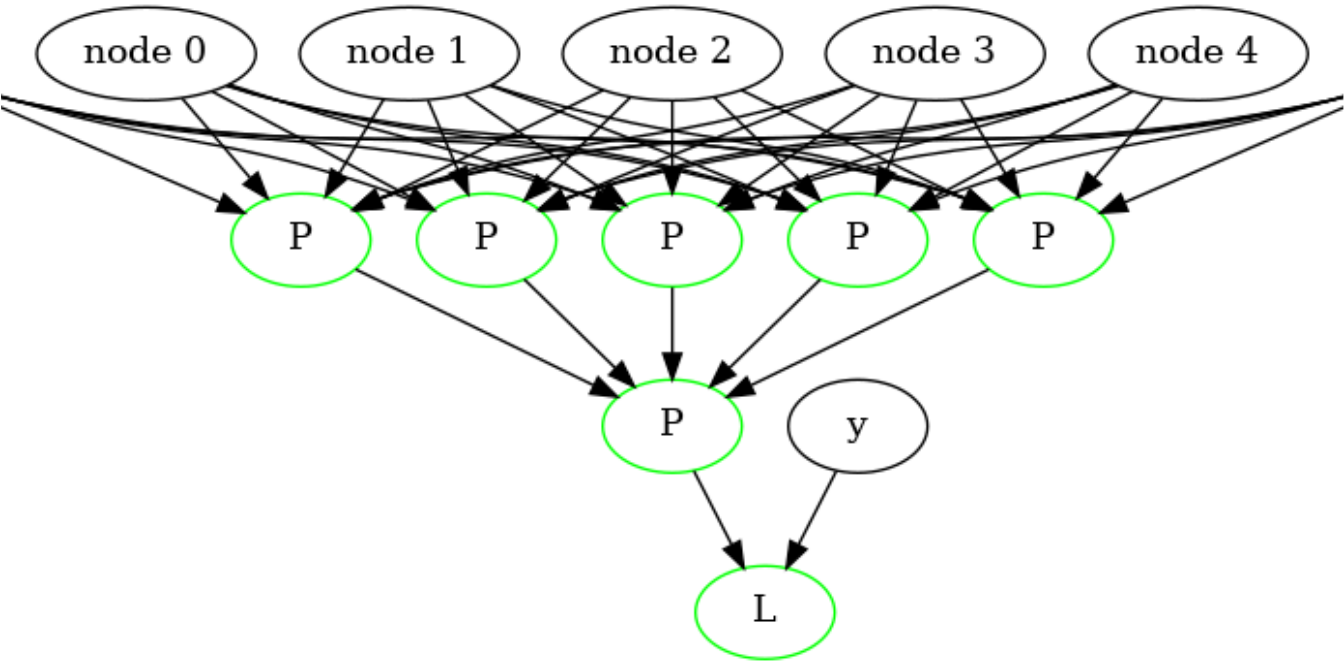
Natural Language Processing

1320 stories · 811 saves



data science and AI

40 stories · 112 saves



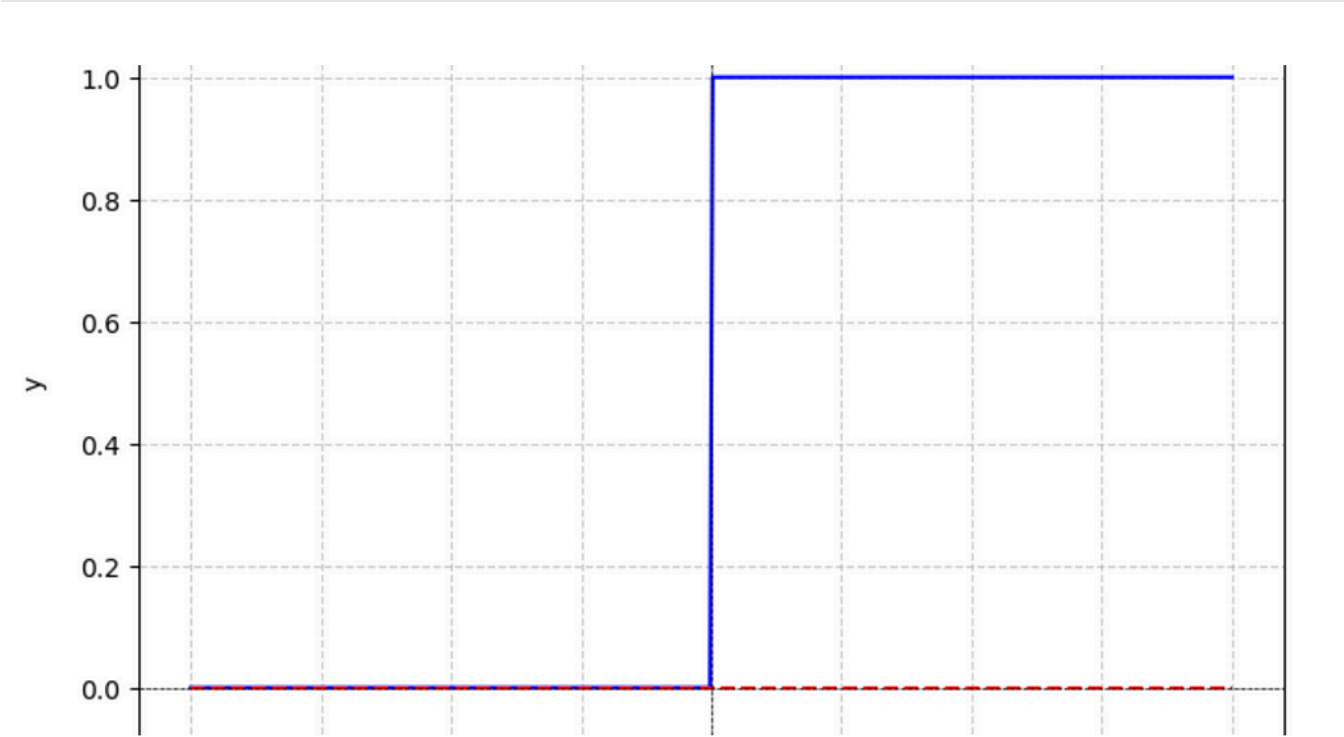
Pranav Joshi


Neural Network from scratch

This isn't one of those articles where you'll be given an overview of the structure of a neural net. Neither is this a "for dummies"...

12 min read · Nov 1, 2023

 145 



 Anushruthika

All about Activation functions & Choosing the Right Activation Function

It is applied to the weighted sum of inputs to the neuron (including the bias term), and the result becomes the output of the neuron that...

18 min read · Nov 9, 2023

 52 



Deep Learning, a subset of machine learning, has revolutionized how we approach complex problems in various fields, from image recognition...

4 min read · Dec 28, 2023



51



See more recommendations