Deep Neural Networks

As redes neurais foram um dos primeiros modelos de aprendizagem de máquina. Sua popularidade já esteve em baixa por duas vezes e agora estamos na terceira onda, graças ao aprendizado profundo. O "profundo" no aprendizado profundo refere-se a uma rede neural com muitas camadas ocultas. Diversos diferentes algoritmos de treinamento, funções de ativação / transferência e estruturas foram adicionados ao longo dos anos. A partir de agora, estudaremos as últimas e mais modernas técnicas de estado da arte para redes neurais profundas.

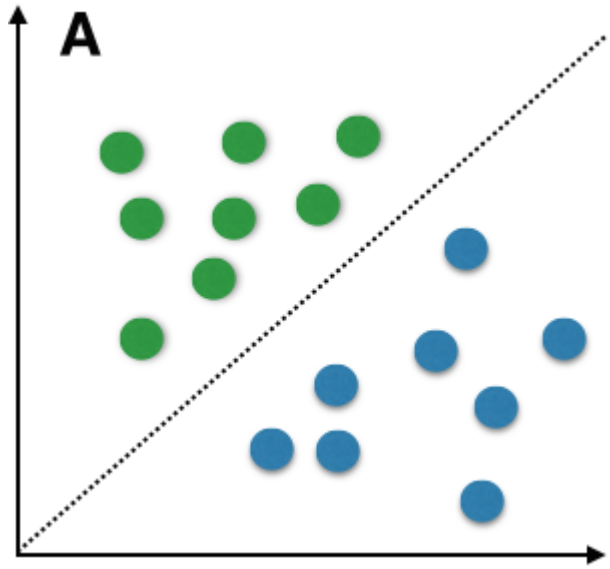
As redes neurais recebem entradas e produzem saída. A entrada para uma rede neural é chamada de vetor de características. O tamanho deste vetor é sempre um comprimento fixo. Alterar o tamanho do vetor de características significa recriar toda a rede neural. Embora o vetor de características seja chamado de "vetor", este não é sempre o caso. Um vetor implica uma matriz 1D. Historicamente, a entrada para uma rede neural foi sempre 1D. No entanto, com redes neurais modernas você pode ter entradas, como:

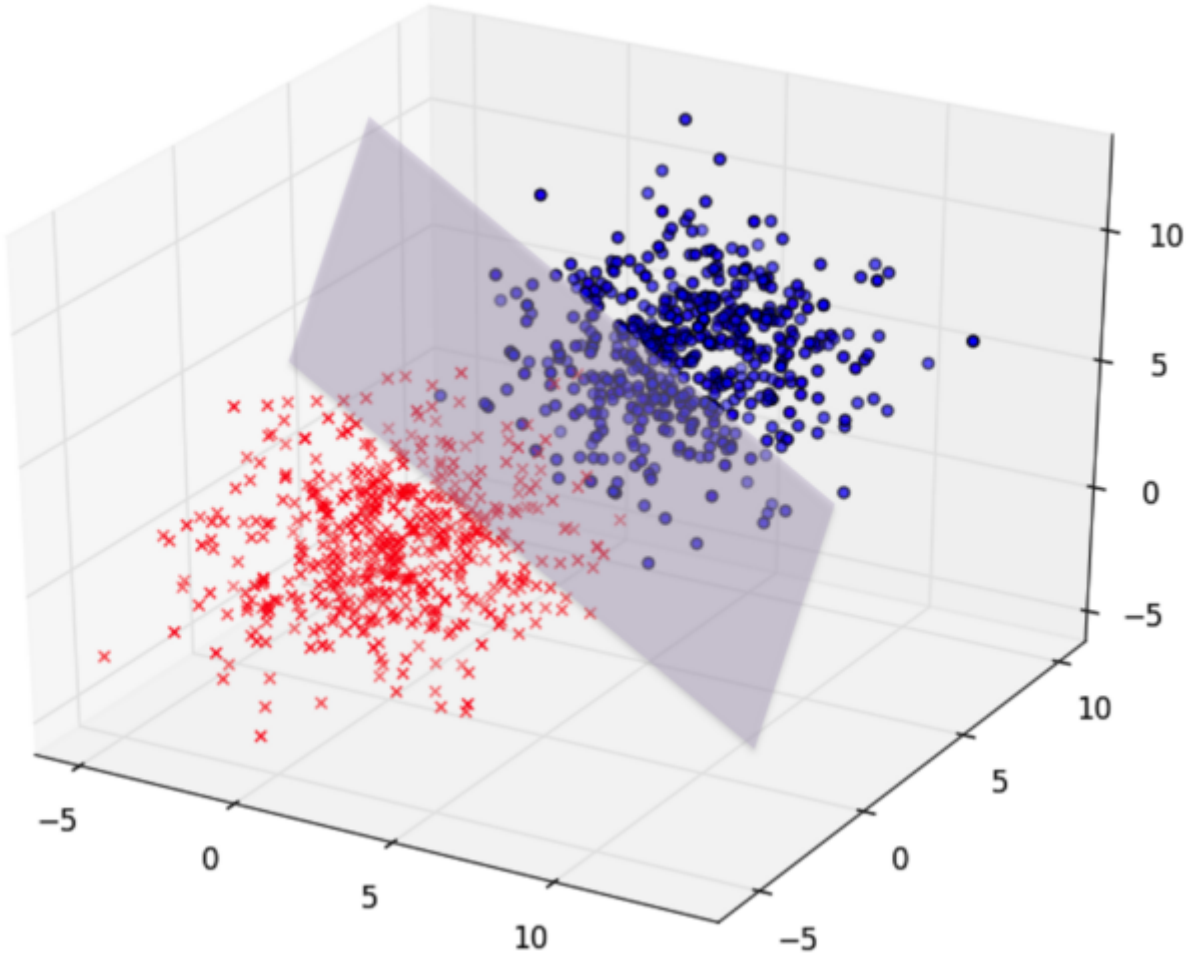
- **1D Vector** - Entrada clássica para uma rede neural, semelhante a linhas em uma planilha. Comum em modelagem preditiva.
- **2D Matrix** - Entrada de imagem em escala de cinza para uma rede neural convolucional (CNN).
- **3D Matrix** - Entrada de imagem colorida para uma rede neural convolucional (CNN).
- **nD Matrix** - Entrada de ordem superior para uma CNN.

Antes da CNN, a entrada de imagens era enviada para uma rede neural simplesmente "achatando" (flatten) a matriz de imagem em uma matriz longa colocando as linhas da imagem lado a lado. As CNNs são diferentes, já que a matriz nD passa literalmente pelas camadas da rede neural.

Inicialmente usaremos objetos 1D para entrada das redes neurais. No entanto, à medida que avançarmos no curso, usaremos objetos de entrada com mais dimensões.

Dimensões O termo dimensão pode ser confuso nas redes neurais. No sentido de um vetor de entrada 1D, a dimensão refere-se a quantos elementos estão na matriz 1D. Por exemplo, uma rede neural com 10 neurônios de entrada tem 10 dimensões. No entanto, agora que temos CNNs, já podemos usar entradas com mais dimensões. A entrada para a rede neural *geralmente* tem 1, 2 ou 3 dimensões, sendo 4 ou mais dimensões incomuns. Você pode ter uma entrada 2D para uma rede neural com 64x64 pixels. Isso resultaria em 4.096 neurônios de entrada. Esta rede é 2D ou 4,096D, dependendo de qual conjunto de dimensões você está falando!


A

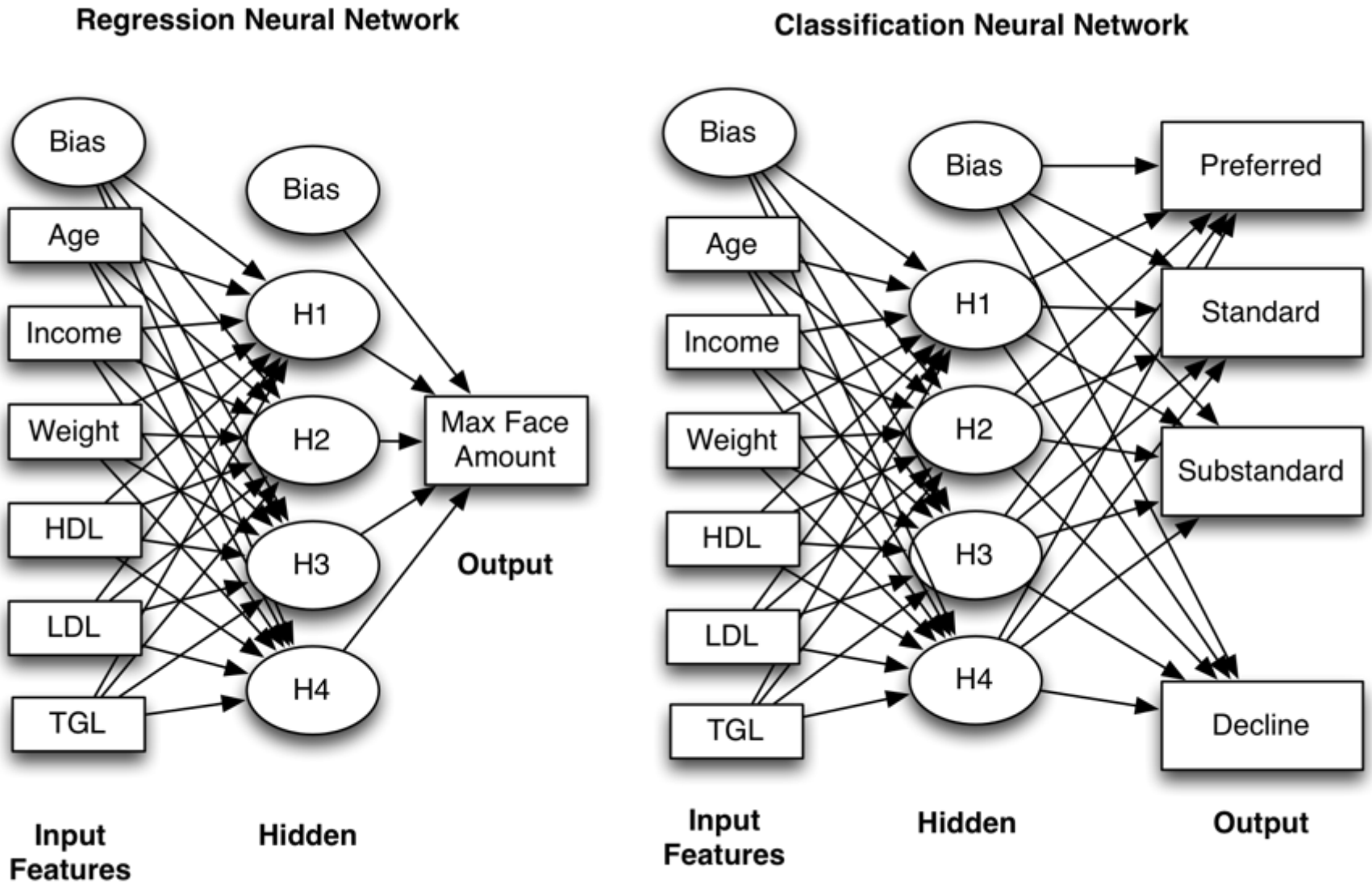

10
5
0
−5
−5
0
5
10
−5
0
5
10

Classificação e Regressão

Como muitos modelos, as redes neurais podem funcionar em classificação ou regressão:

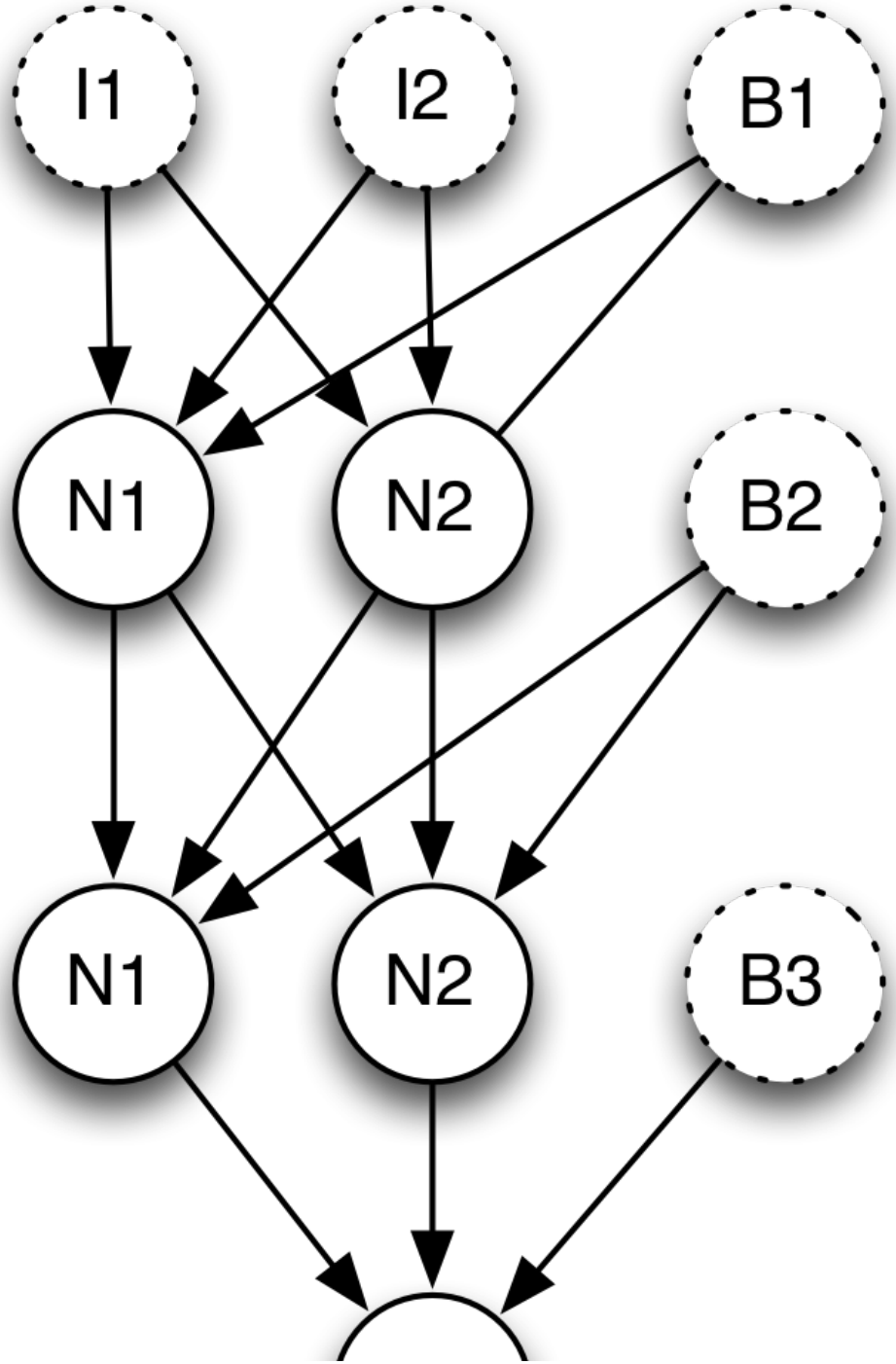
- **Regressão** - Você espera um número como sua previsão da rede neural.
- **Classificação** - Você espera uma classe / categoria como sua previsão da rede neural.

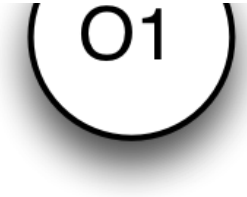
A imagem abaixo mostra uma rede neural de classificação e regressão:



Observe que a saída da rede neural de regressão é numérica e a saída da classificação é uma classe. Seja Regressão ou Classificação de duas classes, as redes sempre têm uma saída única. As redes neurais de classificação possuem um neurônio de saída para cada classe.

O diagrama a seguir mostra uma rede neural típica:


I1
I2
B1
N1
N2
B2
N1
N2
B3
Input Layer
Hidden Layer #1
Hidden Layer #2
Output Layer



Geralmente, existem quatro tipos de neurônios em uma rede neural:

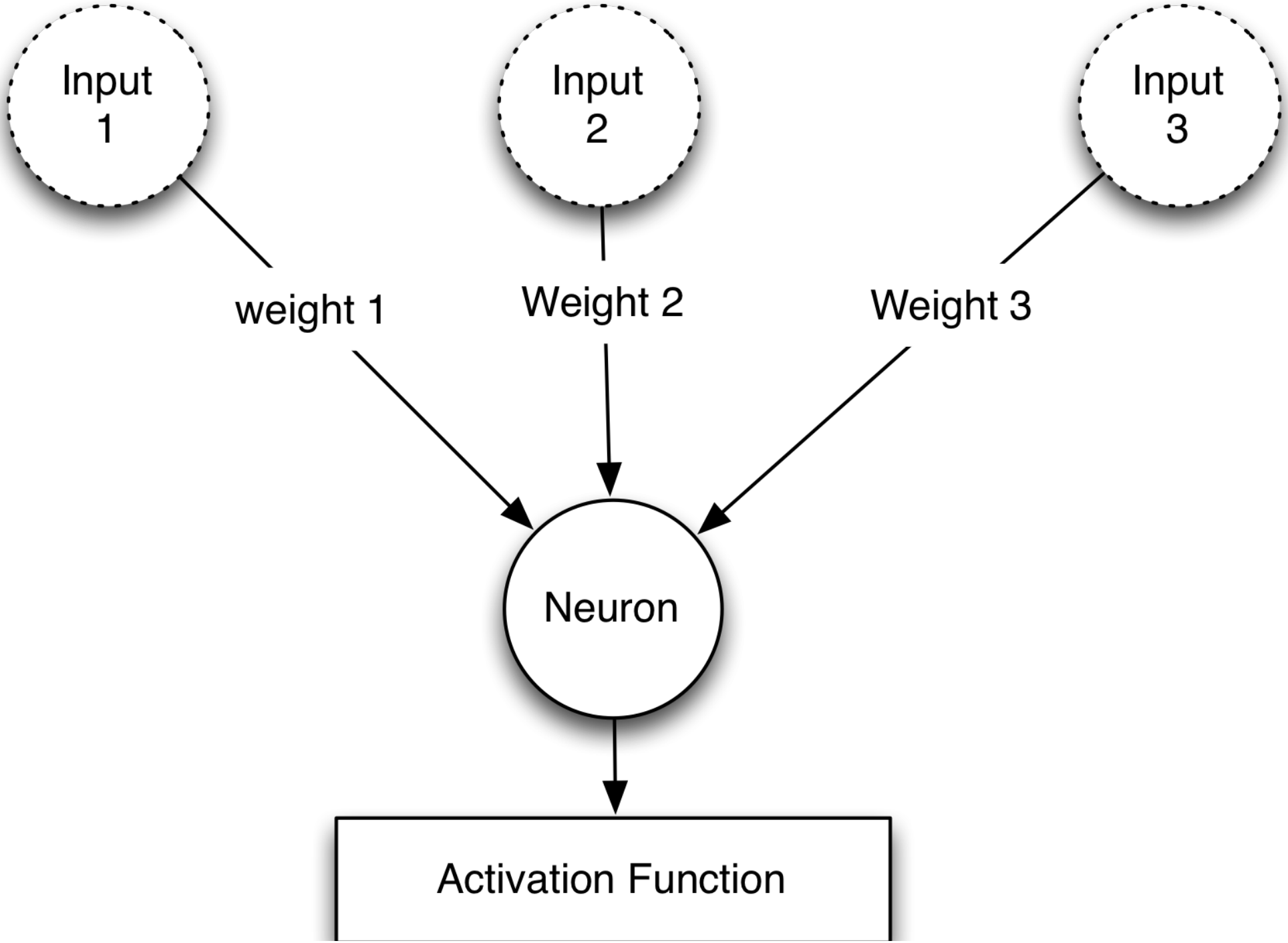
- **Input Neurons** - Cada neurônio de entrada é mapeado para um elemento no vetor de recursos.
- **Hidden Neurons** - Os neurônios ocultos permitem que a rede neural abstraia e processe a entrada na saída.
- **Output Neurons** - Cada neurônio de saída calcula uma parte da saída.
- **Context Neurons** - Mantém estado entre chamadas para a rede neural.
- **Bias Neurons** - Trabalho semelhante ao y-intercepto de uma equação linear.

Cálculo no Neurônio

A saída de um único neurônio é calculada de acordo com a seguinte fórmula:

$f(x, \theta) = \phi(\sum_i (\theta_i \cdot x_i))$

O vetor de entrada (x) representa o vetor de recursos e o vetor $theta$ (theta) representa os pesos. Para dar conta do neurônio de bias, um valor de 1 sempre é anexado ao final do vetor de recursos de entrada. Isso faz com que o último peso seja interpretado como um valor de bias que é simplesmente adicionado à soma. O phi (phi) é a função de transferência / ativação.


Input 1
Input 2
Input 3
weight 1
Weight 2
Weight 3
Neuron
Activation Function

Funções de Ativação

O propósito da função de ativação é introduzir a não linearidade na rede neural. Isso permite que você modele uma variável de resposta que varia de forma não linear com suas variáveis explicativas. Não linear significa que a saída não pode ser reproduzida a partir de uma combinação linear das entradas. Outra maneira de pensar nisso: sem uma função de ativação não-linear na rede, um rede neural, independentemente de quantas camadas tivesse, seria como um perceptron de camada única, porque somar essas camadas lhe daria apenas mais uma função linear.

Uma função de ativação serve como um limite (threshold), alternativamente denominado classificação ou partição. O propósito de uma função de ativação em um contexto de Aprendizagem Profunda (ou seja, várias camadas) é garantir que a representação no espaço de entrada seja mapeada para um espaço diferente na saída. Em todos os casos, uma rede neural realiza uma função de similaridade entre a entrada e os pesos. Este pode ser um produto interno, uma função de correlação ou uma função de convolução. Em todos os casos, é uma medida de semelhança entre os pesos aprendidos e a entrada. Isto é seguido por uma função de ativação que executa um limite na medida de similaridade calculada. No seu sentido mais geral, uma camada de rede neural executa uma projeção que é seguida por uma seleção.

Tanto a projeção quanto a seleção são necessárias para a aprendizagem dinâmica. Sem seleção e apenas projeção, uma rede permanecerá no mesmo espaço e não conseguirá criar níveis mais altos de abstração entre as camadas. A operação de projeção pode, de fato, ser não linear, mas sem a função limiar, não haverá mecanismo para consolidar informações. A operação de seleção impõe a irreversibilidade da informação, um critério necessário para a aprendizagem.

O conceito de uma função de ativação não se aplica realmente aos neurônios humanos. Nos neurônios humanos, a saída consiste em uma série de "faíscas", não um único valor determinado por uma função.

As funções de ativação, também conhecidas como funções de transferência, são usadas para calcular a saída de cada camada de uma rede neural. Historicamente, as redes neurais utilizaram uma função hiperbólica tangente, sigmoide / logística ou linear. No entanto, as redes neurais profundas modernas utilizam principalmente as seguintes funções de ativação:

- **Rectified Linear Unit (ReLU)** - Usado para a saída de camadas ocultas.
- **Softmax** - Usado para a produção de redes neurais de classificação.
- **Linear** - Usado para a produção de redes neurais de regressão (ou classificação de 2 classes).

A função ReLU é calculada da seguinte forma:

$$\phi(x) = \max(0, x)$$

O Softmax é calculado da seguinte forma:

$$\phi_i(z) = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}}$$

A função de ativação Softmax é útil apenas com mais de um neurônio de saída. Ela garante que todos os neurônios de saída somem 1,0. Isso torna muito útil para a classificação onde mostra a probabilidade de cada uma das classes como sendo a escolha correta.

A função de ativação linear é essencialmente uma função de ativação usada para problemas de regressão:

$$\phi(x) = x$$

A função de ativação determina a saída gerada por um nó com base em sua entrada. As funções de ativação Sigmoid foram muito populares e a ReLU é atualmente muito popular. A função de ativação é definida no nível da camada e se aplica a todos os neurônios nessa camada.

Aqui uma lista das principais funções de ativação usads atualmente!

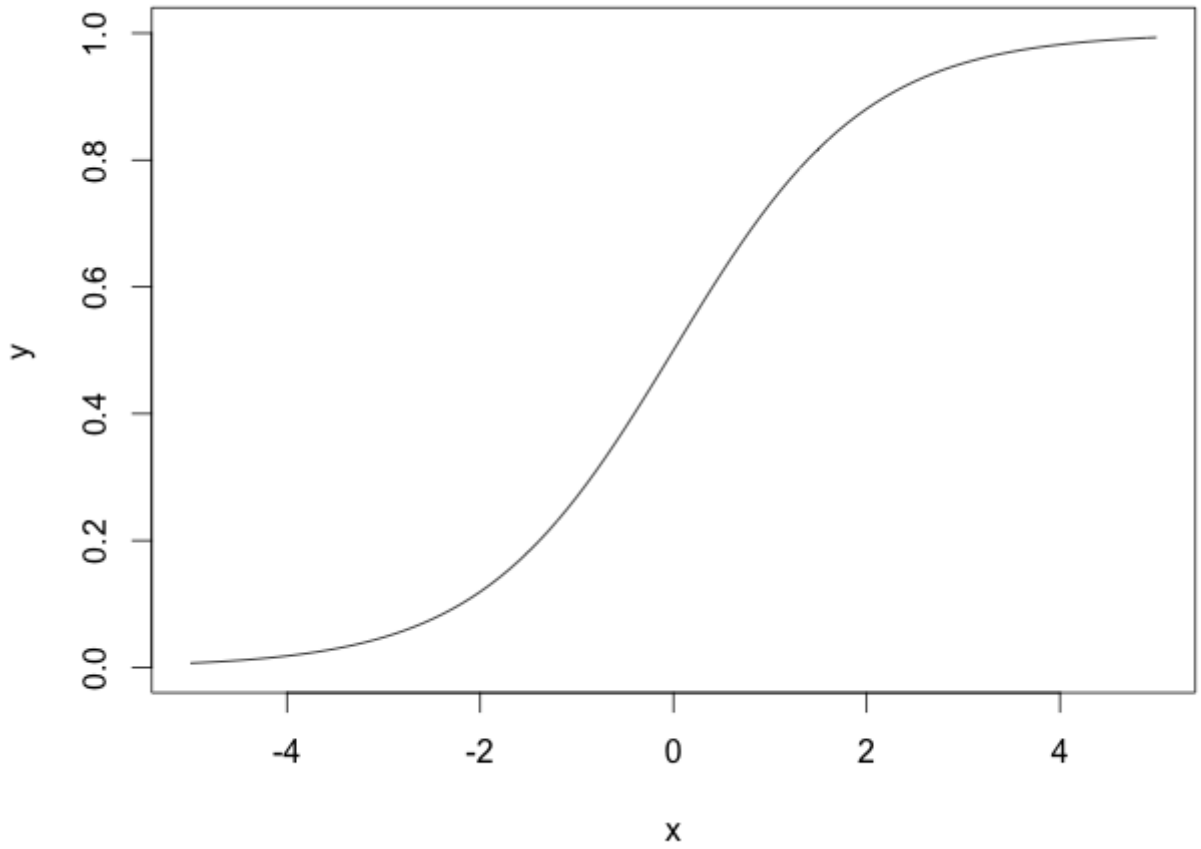
- CUBE
- ELU
- HARDSIGMOID
- HARDTANH
- IDENTITY
- LEAKYRELU
- RATIONALTANH
- RELU
- RRELU
- SIGMOID
- SOFTMAX
- SOFTPLUS
- SOFTSIGN
- TANH

O propósito original da função de ativação na rede neural multicamada é separar a transformação linear múltipla sucessiva pela não-linearidade, caso contrário, elas colapsarão para uma única transformação linear.

Supondo que, "recomendação positiva cria ambiente positivo", a maior parte das funções de ativação suprime o valor negativo. Além disso, o requisito de diferenciação vem do procedimento de treinamento baseado em gradiente. Mais recentemente, para escapar da saturação, a ReLU é a função de ativação tem sido a mais promissora entre pesquisadores.

Função Sigmóide

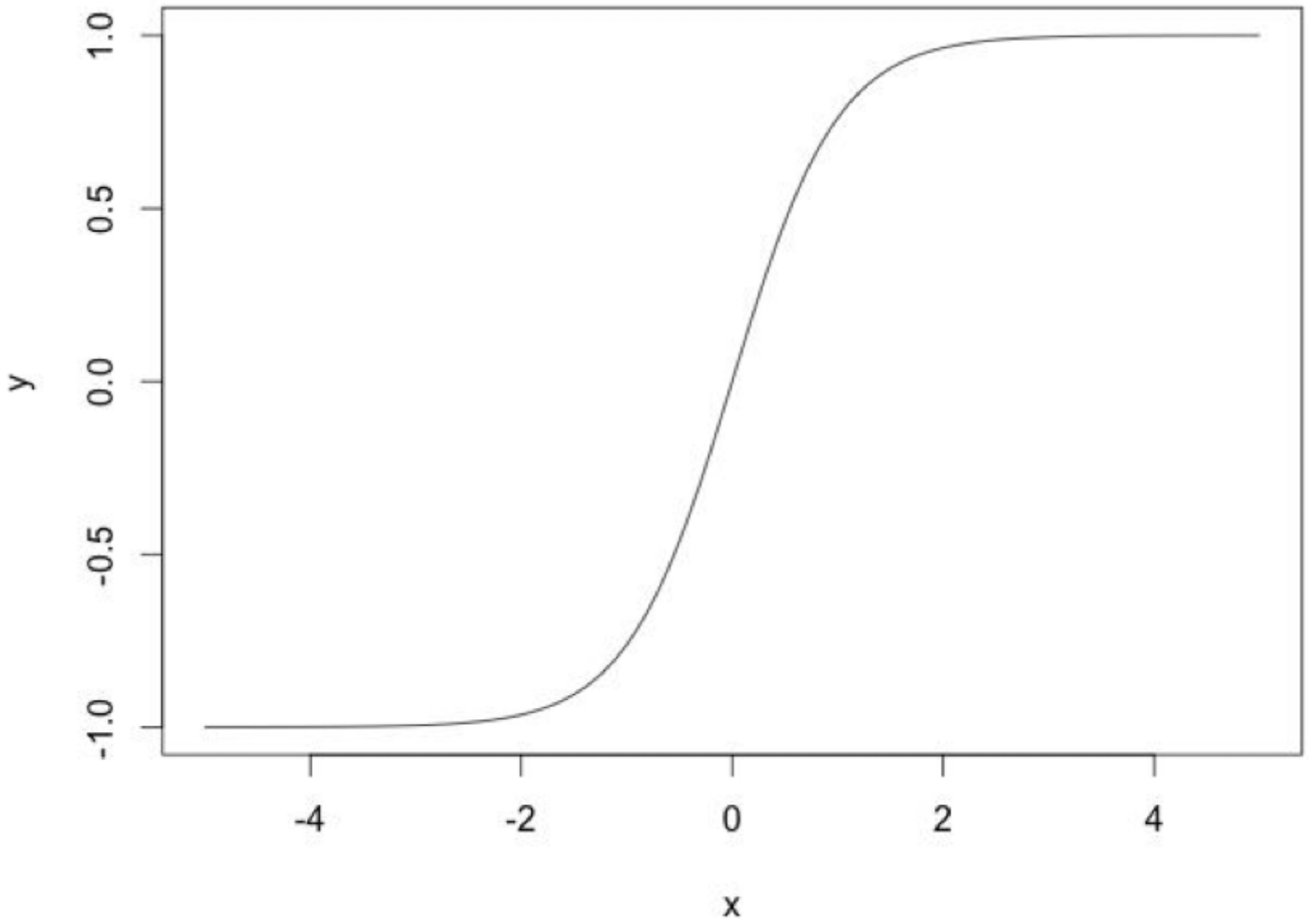
A função de ativação sigmóide ou logística é uma escolha muito comum para redes neurais feedforward que precisam produzir apenas números positivos. Apesar do seu uso generalizado, a função de ativação tangente hiperbólica ou de unidade linear rectificada (ReLU) geralmente é uma escolha mais adequada. Usamos a função sigmoid para garantir que os valores permaneçam dentro de uma faixa relativamente pequena.



Como você pode ver no gráfico acima, os valores de x acima ou abaixo de 0 são compactados para o intervalo aproximado entre 0 e 1.

Função Hiperbólica Tangente

A função hiperbólica tangente também é uma função de ativação muito comum para redes neurais que devem produzir valores no intervalo entre -1 e 1. Essa função de ativação é simplesmente a função tangente hiperbólica (tanh). O gráfico da função hiperbólica tangente tem uma forma semelhante à função de ativação sigmóide:

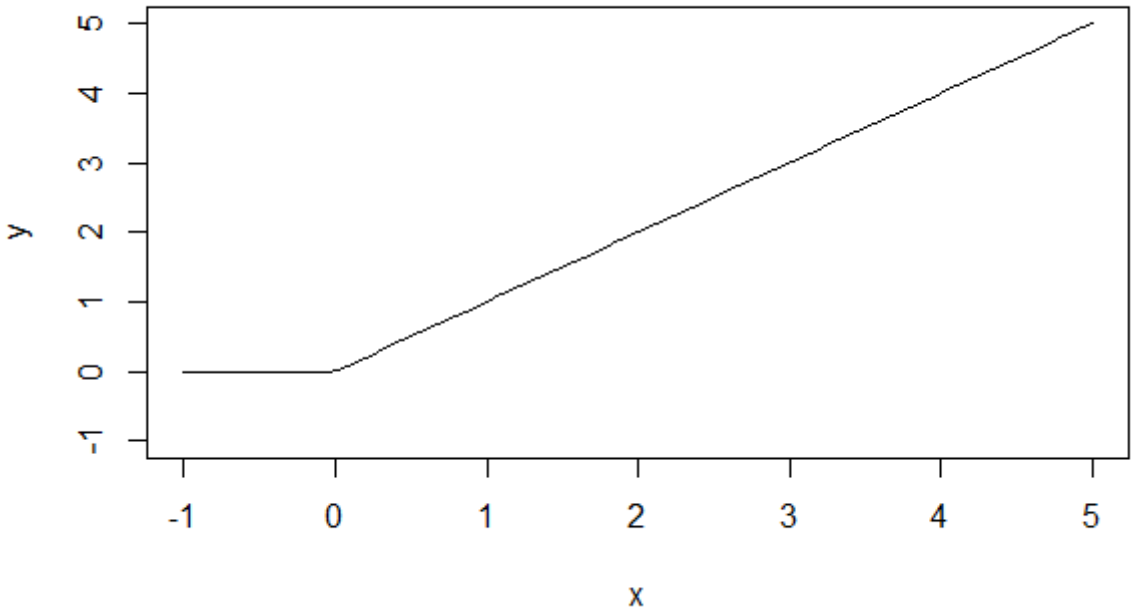


A função tangente hiperbólica possui várias vantagens em relação à função de ativação sigmoide, que envolvem as derivadas utilizadas no treinamento da rede neural. Vamos explorar essas vantagens mais a frente!

Por que a ReLU?

Introduzido em 2000, a unidade linear retificada (ReLU) teve uma adoção muito rápida ao longo dos últimos anos. Antes da função de ativação ReLU, a hiperbólica tangente era geralmente aceita como função de ativação principal para modelos de redes neurais profundas. A pesquisa mais atual agora recomenda a ReLU devido a resultados de treinamento superiores. Como resultado, a maioria das redes neurais deve utilizar a ReLU em camadas ocultas e softmax ou linear na camada de saída.

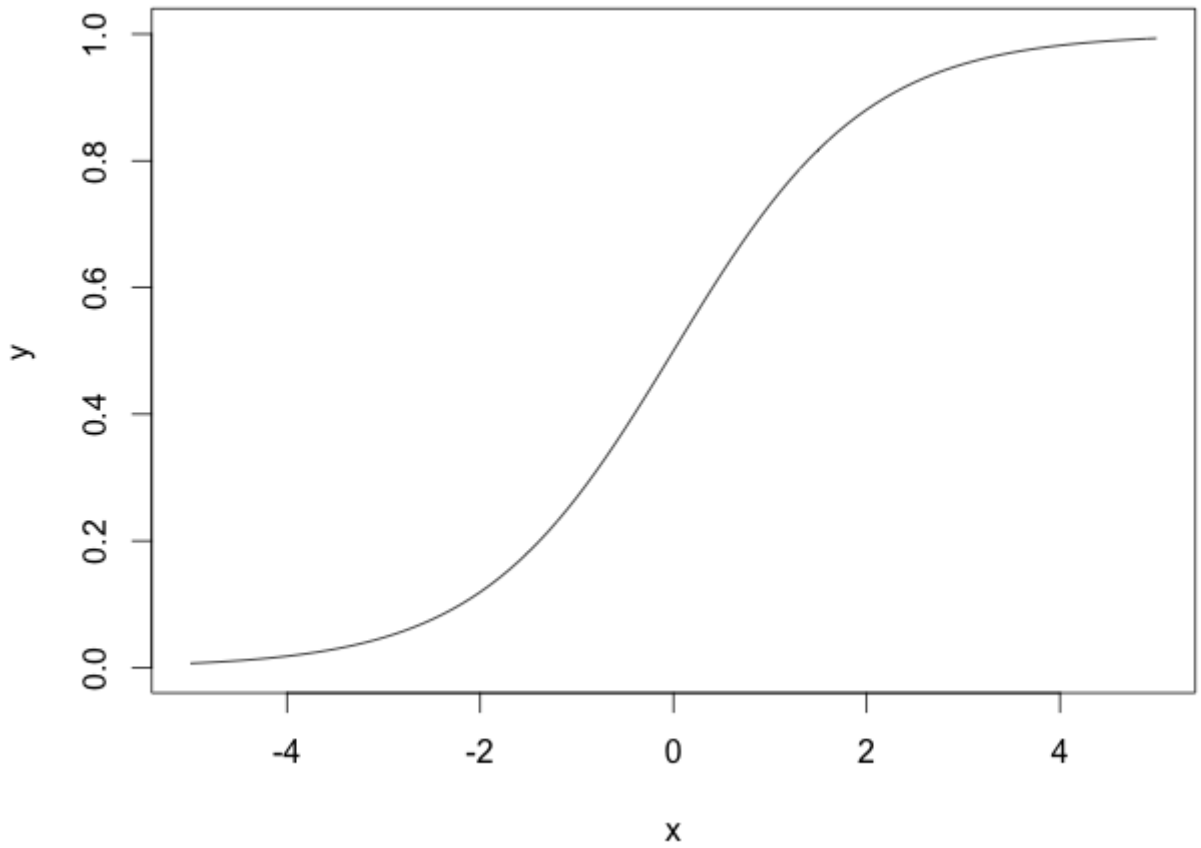
Vamos agora examinar por que a ReLU normalmente funciona melhor do que outras funções de ativação para camadas ocultas. Parte do desempenho deve-se ao fato de que a função de ativação ReLU é uma função linear e não saturadora. Ao contrário das funções de ativação sigmoide ou hiperbólica, a ReLU não satura para -1, 0 ou 1. Uma função de ativação de saturação move-se e, eventualmente, atinge um valor. A função hiperbólica, por exemplo, satura para -1 à medida que x diminui e para 1 quando x aumenta. Mas observe o comportamento da ReLU:



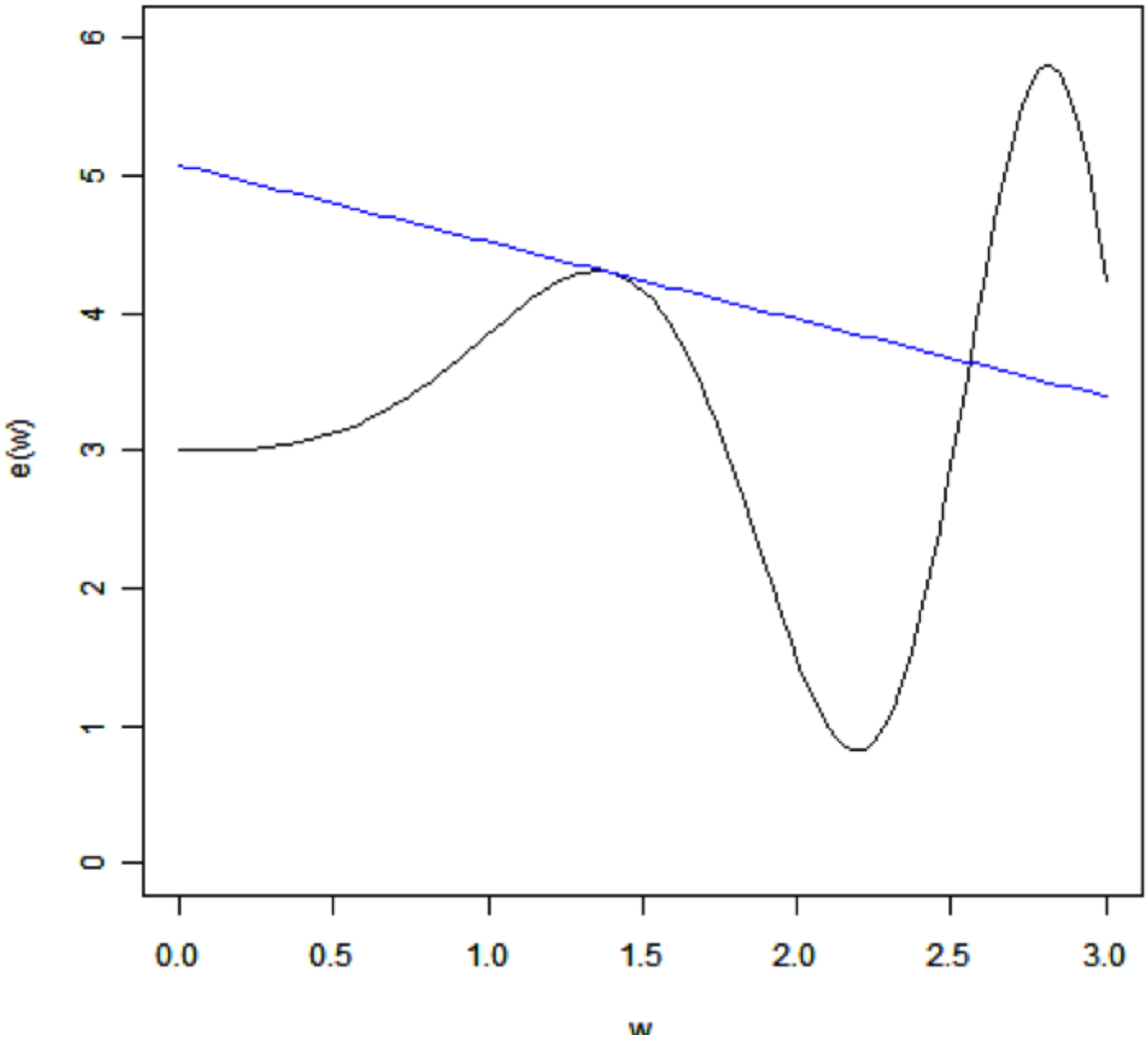
Por que a função de ativação ReLU é tão popular? Foi uma das principais melhorias nas redes neurais que faz o aprendizado profundo realmente funcionar. Antes da aprendizagem profunda, a função de ativação sigmoide era muito comum:

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

O gráfico da função sigmoide é mostrado aqui:



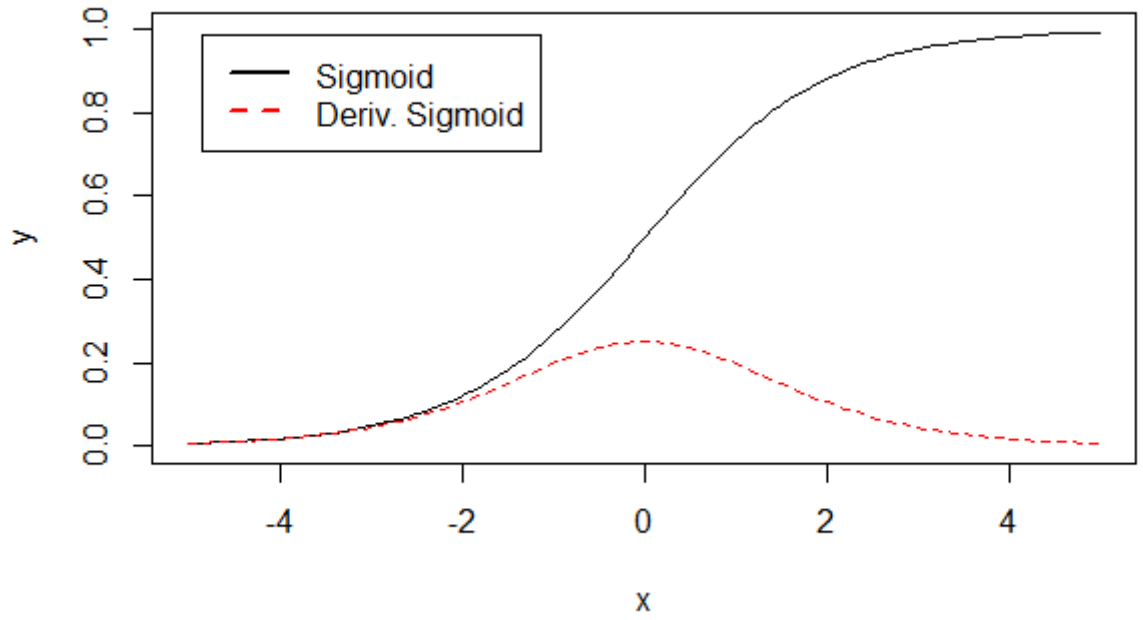
As redes neurais são frequentemente treinadas usando descida do gradiente. Para fazer uso da descida gradiente, é necessário tomar a derivada da função de ativação. Isso permite que as derivadas parciais de cada um dos pesos sejam calculadas em relação à função de erro. Uma derivada é a taxa de mudança instantânea:


e(w)
0
1
2
3
4
5
6
0.0
0.5
1.0
1.5
2.0
2.5
3.0
w

A derivada da função sigmoide é dada aqui:

$$\phi'(x) = \phi(x)(1 - \phi(x))$$

O gráfico da derivada sigmoide é dado aqui:



A derivada rapidamente satura para zero à medida que *x* se move de zero. Este não é um problema para a derivada da ReLU, que é dada aqui:

$$\phi'(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

Ou seja, valores negativos são transformados em zero!

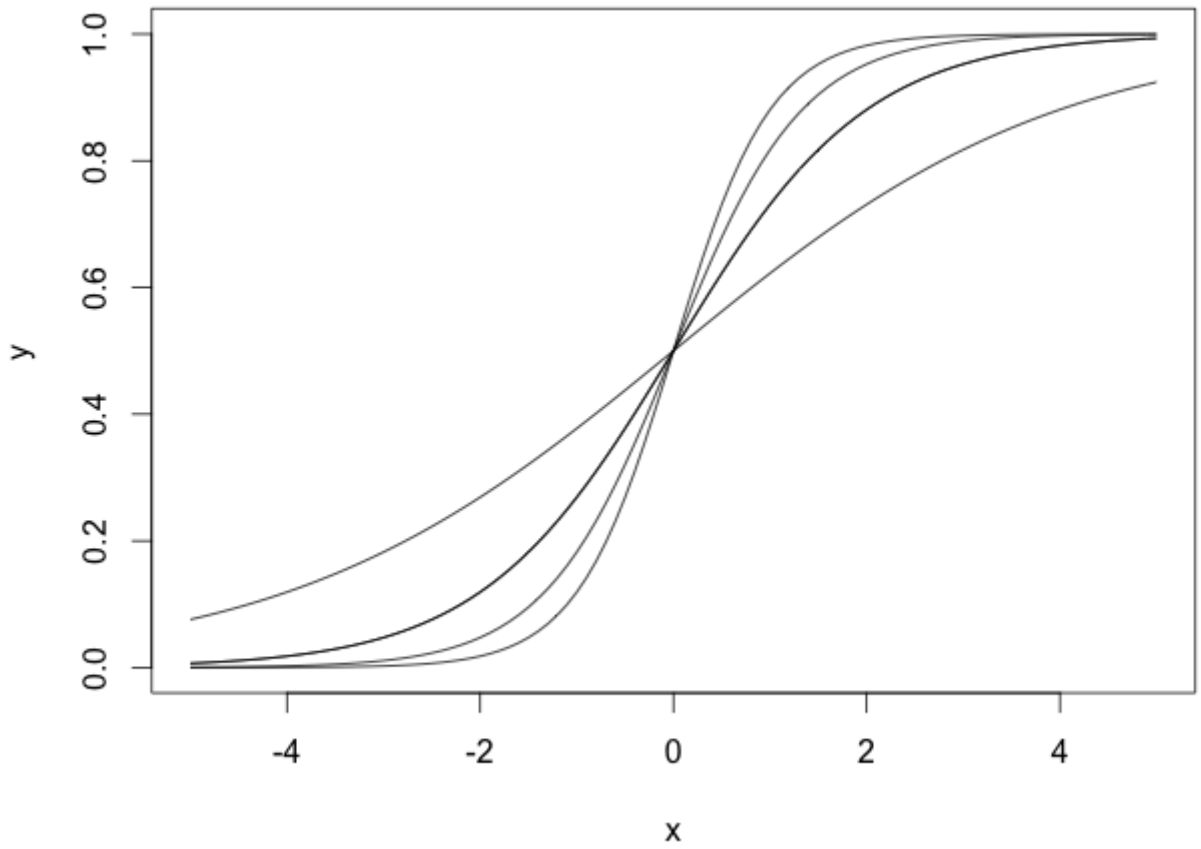
Por Que Precisamos do Bias?

As funções de ativação observadas na seção anterior especificam a saída de um único neurônio. Juntos, o peso e o bias de um neurônio moldam a saída da ativação para produzir a saída desejada. Para ver como esse processo ocorre, considere a seguinte equação. Representa uma rede neural de ativação sigmoid de entrada única.

$$f(x, w, b) = \frac{1}{1+e^{-(wx+b)}}$$

A variável *x* representa a entrada única para a rede neural. As variáveis *w* e *b* especificam o peso e o bias da rede neural. A equação acima é uma combinação da soma ponderada das entradas e da função de ativação sigmoide. Para esta seção, consideraremos a função sigmoide porque demonstra claramente o efeito de um neurônio de bias.

Os pesos do neurônio permitem ajustar a inclinação ou a forma da função de ativação. A figura a seguir mostra o efeito na saída da função de ativação sigmóide se o peso for variado:

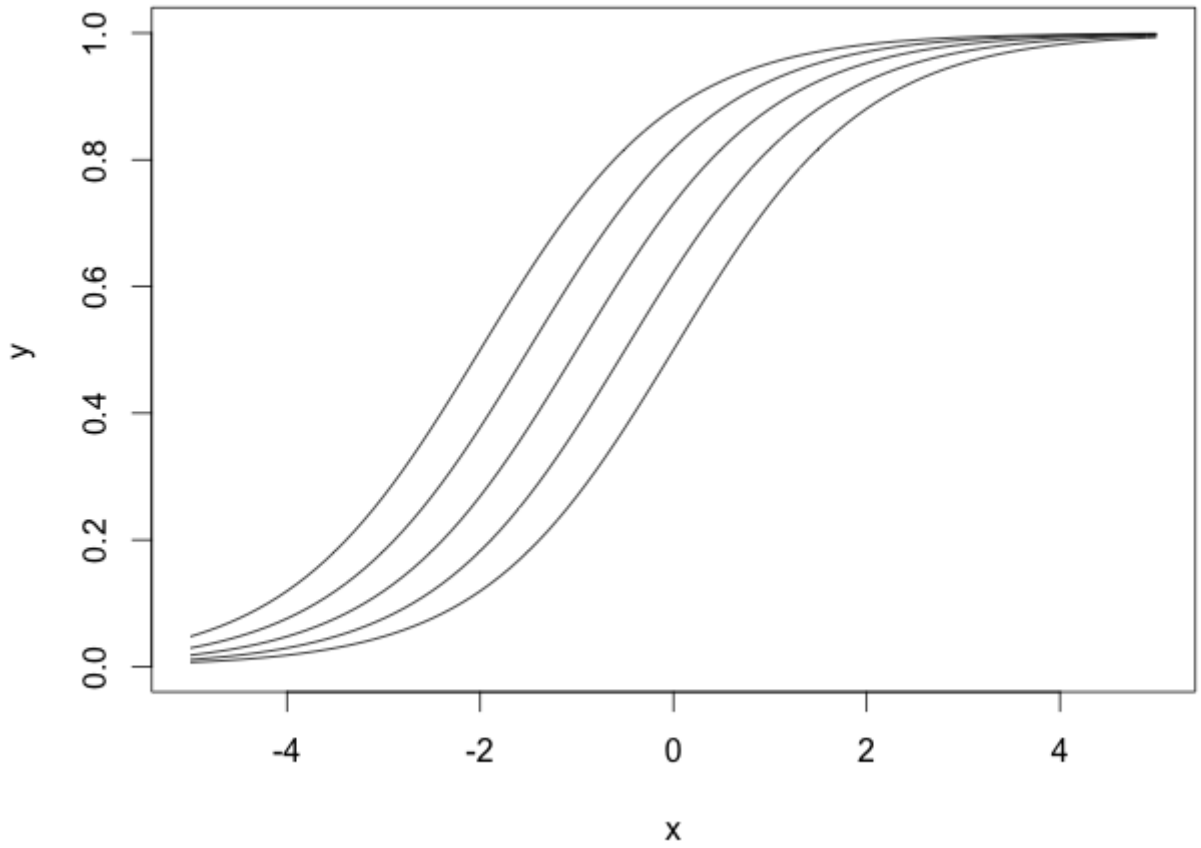


O diagrama acima mostra várias curvas sigmoides usando os seguintes parâmetros:

```
f(x,0.5,0.0)
f(x,1.0,0.0)
f(x,1.5,0.0)
f(x,2.0,0.0)
```

Para produzir as curvas, não utilizamos o bias, o que é evidente no terceiro parâmetro igual a 0 em cada caso. Usando quatro valores de peso produz quatro curvas sigmóides diferentes na figura acima. Não importa o peso, sempre obtemos o mesmo valor de 0,5 quando x é 0 porque todas as curvas atingem o mesmo ponto quando x é 0. Podemos precisar da rede neural para produzir outros valores quando a entrada é próxima de 0,5.

O Bias desloca a curva sigmóide, que permite valores diferentes de 0,5 quando x está próximo de 0. A figura a seguir mostra o efeito de usar um peso de 1,0 com diferentes valores de bias:



O diagrama acima mostra várias curvas sigmoides com os seguintes parâmetros:

```
f(x,1.0,1.0)
f(x,1.0,0.5)
f(x,1.0,1.5)
f(x,1.0,2.0)
```

Utilizamos um peso de 1.0 para estas curvas em todos os casos. Quando utilizamos diferentes tipos de bias, as curvas sigmoidais foram deslocadas para a esquerda ou para a direita. Como todas as curvas se fundem no canto superior direito ou inferior esquerdo, não é uma mudança completa.

softmax function

In [1]:
```
import numpy as np

def softmax(x):
    """
    Compute softmax values for each sets of scores in x.

    Rows are scores for each class.
    Columns are predictions (samples).
    """
    scoreMatExp = np.exp(np.asarray(x))
    return scoreMatExp / scoreMatExp.sum(0)

l = [0.9, 0.4, 0.2]
print(softmax(l))
```

```
[0.47548496 0.2883962  0.23611884]
```