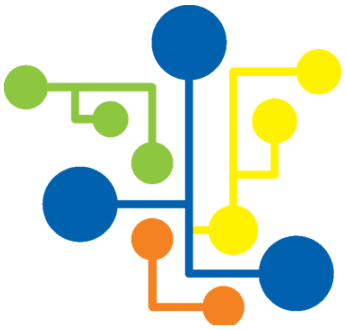


Data Science Academy

www.datascienceacademy.com.br

Matemática Para Machine Learning

Estudo de Caso 3
Transformação Linear em Redes Neurais Artificiais

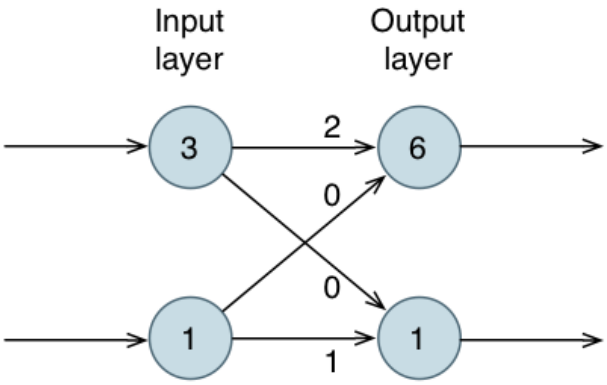
Nós já sabemos que a Álgebra Linear nos oferece ferramentas para trabalhar com transformações lineares, tais como:

- Matrizes que representam transformações lineares.
- Vetores que representam pontos individuais.
- Aplicação de uma transformação linear a um ponto ou conjunto de pontos usando multiplicação de matrizes.
- Composição de transformações lineares usando multiplicação de matrizes.

Acontece que as redes neurais artificiais oferecem outra estrutura para fazer a mesma coisa (uma rede neural artificial nada mais é do que uma conjunto de operações matemáticas). Digamos que queremos implementar uma transformação linear que use um vetor (x, y) e produza (2x, y). Na álgebra linear, faríamos algo assim (usando (3, 1) como exemplo para x e y respectivamente):

$$\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 6 \\ 1 \end{bmatrix}$$

Em uma rede neural, faríamos isso:



No diagrama de rede neural acima, cada unidade de saída produz a combinação linear das entradas e dos pesos de conexão, que é a mesma coisa que fazemos com a multiplicação de matrizes.

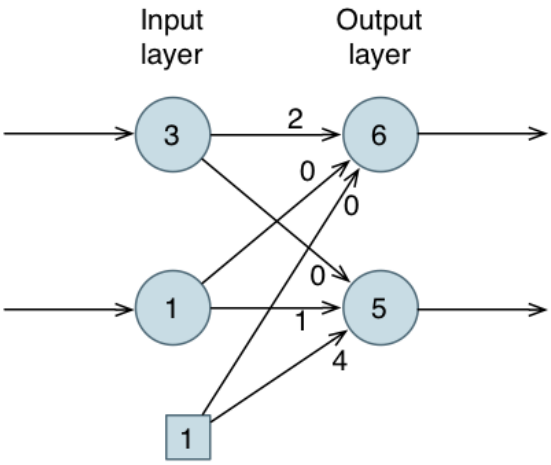
Mas há uma característica particular em nosso exemplo de rede neural. A maioria dos fenômenos que queremos modelar envolvem relacionamentos não-lineares e não podemos "espremer" a não linearidade de transformações lineares. Vamos ver como podemos ajustar nossa rede neural para resolver esses problemas.

Vamos continuar com os exemplos que usamos acima. Com a álgebra linear, geralmente lidamos com transformações afins usando a adição de vetores:

$$\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 4 \end{bmatrix} = \begin{bmatrix} 6 \\ 5 \end{bmatrix}$$

(Há outra maneira de fazer isso usando matrizes aumentadas, mas usar a adição vetorial é mais comum).

Como podemos fazer isso com uma rede neural? O truque é algo chamado de entradas de polarização (bias em inglês ou ainda viés em português):



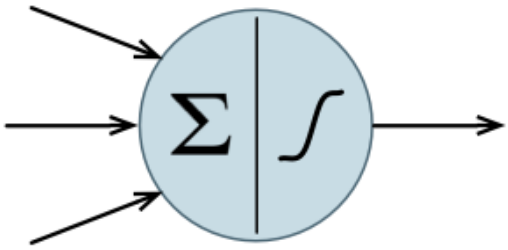
Cada unidade de saída executa uma combinação linear dos pesos (valores numéricos que são aprendidos no treinamento da rede) e entradas (os dados propriamente ditos) recebidos. Desta vez, porém, as unidades têm uma entrada de polarização constante (no diagrama o quadrado com o número um), que cada unidade de saída pode pesar de forma independente para alcançar o efeito de um vetor. Neste caso, usamos o peso 0 para a primeira unidade de saída para zerar a tendência. Usamos o peso 4 para a segunda unidade de saída para dimensionar a tendência de acordo.

A propósito, elas são chamadas de entradas de polarização porque a entrada ponderada determina o valor da linha de base para a unidade. Você pode pensar nisso como sendo o termo b na equação y = mx + b padrão da regressão linear.

Introduzindo a não linearidade usando funções de ativação

Agora temos transformações afins, mas nunca vamos construir robôs do tipo Terminator se pararmos por aí. Precisamos de alguma maneira de introduzir a não-linearidade real.

Para isso usamos a função de ativação. A ideia é que podemos dotar cada unidade com um pouco de pós-processamento, executando a combinação linear através de uma função de ativação não-linear e tratando-a como a saída da unidade. Se parece com isso:

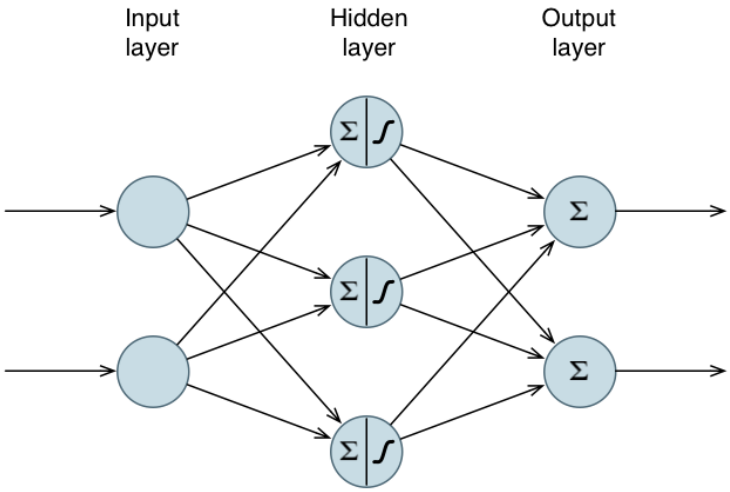


Há diversas opções diferentes disponíveis para a função de ativação. A mais comum era a função Sigmóide, mas hoje em dia as pessoas usam diferentes funções de ativação para diferentes propósitos. Usar uma função de ativação não-linear é o que nos permite escapar da linearidade e assim usar o algoritmo para resolver os mais variados tipos de problemas. A vida não é linear (você deve saber disso) e para modelar os problemas do mundo real, primeiro aplicamos transformações lineares e depois ajustamos com ativações não-lineares. Paradoxal, mas necessário.

Arquiteturas Multicamadas

Acontece que, para fazer qualquer coisa interessante, queremos ter pelo menos uma camada oculta de unidades posicionadas entre as camadas de entrada e saída. Isso possibilita aprender funções mais complexas, sendo o exemplo clássico, o XOR da lógica (ou exclusivo).

De qualquer forma, aqui está o que uma rede feedforward com pelo menos uma camada oculta (tal rede também é conhecida como Perceptron Multicamada) se parece:



Sob certas suposições razoavelmente gerais, ter uma única camada oculta é suficiente para transformar nossa rede feedforward em um aproximador universal.

Com o aprendizado profundo (Deep Learning), a ideia é ter muitas camadas ocultas. De uma perspectiva teórica, elas não são necessários, mas, na prática, elas fazem toda a diferença. Cada camada sucessiva fornece suporte para representações mais abstratas do que aquelas que vivem na camada anterior. Por exemplo, sua camada de entrada pode ter vários pixels. Em seguida, sua próxima camada representa as bordas e a próxima representa as formas, e assim por diante. No momento em que você chegar às camadas finais, você pode ter representações para coisas como um gato ou um automóvel.

Equipe DSA