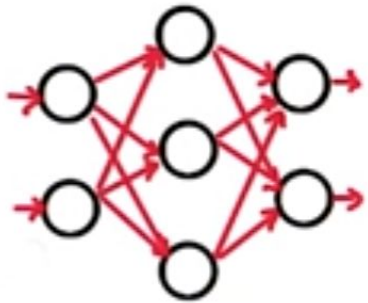
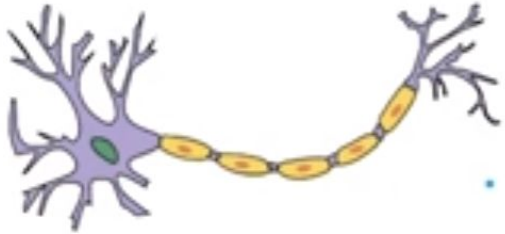
Redes Neurais Artificiais - 3



NEURONS?

HOW THE BRAIN WORKS?



NEUROMORPHIC ENGINEERING?

Perceptron Multicamadas - MPL

Implementando a camada oculta

Pré-requisitos

Abaixo, detalharemos a matemática das redes neurais em um perceptron multi-camada. Com múltiplos perceptrons, nós iremos utilizar vetores e matrizes. Para revisar estes tópicos, dê uma olhada em:

1. Introdução aos vetores da Khan Academy.
2. Introdução às matrizes da Khan Academy.

Perceptron Multicamadas - MPL

Cont… Implementando a camada oculta

Para inicializar esses pesos usando o Numpy, temos que fornecer o formato da matriz. Caso a variável `features` seja um vetor de duas dimensões contendo os dados de entrada:

```
# Número de observações e unidades de input
n_observacoes, n_inputs = features.shape
# Número de unidades ocultas
n_ocultas = 2
pesos_inputs_para_ocultas = np.random.normal(0, n_inputs**-0.5, size=(n_inputs, n_ocultas))
```

Isso cria um vetor 2D (ou seja, uma matriz) chamado `pesos_inputs_para_ocultas` com as dimensões `n_inputs` por `n_ocultas`. Lembre-se de que o input de uma unidade oculta é a soma de todos os inputs multiplicado pelos pesos da unidade oculta.

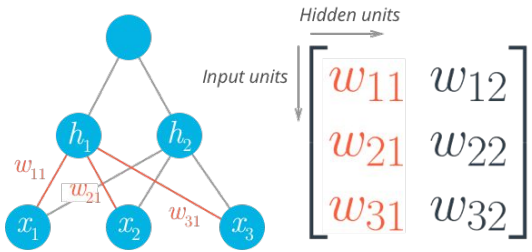
Perceptron Multicamadas - MPL

Implementando a camada oculta

Derivação

Antes, estávamos lidando apenas com um nó de output, o que simplifica o código. No entanto, agora que temos múltiplas unidades de input e múltiplas unidades ocultas, os pesos entre elas precisam de dois índices: w_{ij} sendo que i indica as unidades de input e j as unidades ocultas.

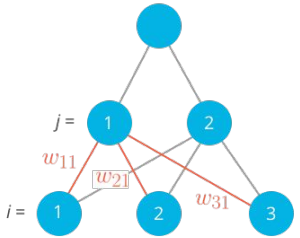
Por exemplo, a imagem a seguir mostra a nossa rede, com as unidades de input marcadas como x_1,x_2 e x_3 assim como as unidades ocultas marcadas como h_1 e h_2:



Perceptron Multicamadas - MPL

Cont… Implementando a camada oculta

Lembre-se de que o input de uma unidade oculta é a soma de todos os inputs multiplicado pelos pesos da unidade oculta. Então para cada unidade da camada oculta h_j, calcularemos o seguinte: $h_j = \sum_i w_{ij} x_i$



$$\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \times \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}$$

Perceptron Multicamadas - MPL

Cont… Implementando a camada oculta

Para o input da segunda unidade da camada oculta, se calcula o produto escalar dos inputs com a segunda coluna da matriz de pesos. Esse padrão continua.

Com o Numpy, é possível fazer isso para todos os inputs e todos os outputs de uma vez usando `np.dot`

```
inputs_ocultos = np.dot(inputs, pesos_inputs_para_ocultas)
```

Você pode definir a matriz de pesos de modo que ela tenha as dimensões `n_ocultas` por `n_inputs` e então multiplicar de modo que os inputs formem um *vetor coluna*:

$$h_j = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Para uma melhor notação e mantendo os mesmos valores dos pesos, porém trocando os índices. Tenha em mente que essa é a mesma matriz de peso de antes, mas com um giro de modo que a primeira coluna agora é a primeira linha, a segunda coluna agora é a segunda linha.

Perceptron Multicamadas - MPL

Cont… Implementando a camada oculta

Criando um vetor coluna

Vimos antes que as vezes o desejado é ter um vetor coluna, ainda que os vetores Numpy, por padrão, funcionem como vetores linha. É possível fazer a transposta de qualquer vetor usando `arr.T`, mas para um vetor de uma dimensão, a transposta será um vetor linha. Ao invés disso, use `arr[:,None]` para criar um vetor coluna:

```
print(features)
> array([ 0.49671415, -0.1382643 ,  0.64768854])

print(features.T)
> array([ 0.49671415, -0.1382643 ,  0.64768854])

print(features[:, None])
> array([[ 0.49671415],
       [-0.1382643 ],
       [ 0.64768854]])
```

Perceptron Multicamadas - MPL

Cont… Implementando a camada oculta

Outra opção é criar vetores com duas dimensões, então usar `arr.T` para obter o vetor coluna.

```
np.array(features, ndmin=2)
> array([[ 0.49671415, -0.1382643 ,  0.64768854]])

np.array(features, ndmin=2).T
> array([[ 0.49671415],
       [-0.1382643 ],
       [ 0.64768854]])
```

Tarefa 4 - MPL 4x3x2

A seguir, você implementará uma rede 4x3x2 orientada a frente, com funções de ativação sigmóide em ambas as camadas.

Coisas a fazer:

- Calcular o input da camada oculta.
- Calcular o output da camada oculta.
- Calcular o input da camada de output.
- Calcular o output da rede.

Perceptron Multicamadas - MPL

Backpropagation

Agora chegamos ao problema de como fazer uma rede neural de múltiplas camadas *aprender*. Antes, vimos como atualizar os pesos usando o gradiente descendente. O algoritmo de Retropropagação (backpropagation daqui em diante) é apenas uma extensão disso, usando a regra de cadeia para encontrar o erro respeitando os pesos conectando a camada de input para a camada oculta (numa rede de duas camadas).

Para atualizar os pesos das camadas ocultas usando o gradiente descendente, é necessário saber quanto contribuiu cada unidade oculta para a produção daquele erro no output final. Uma vez que o output de uma camada é determinado pelos pesos entre camadas, o erro resultante de unidades é proporcional aos pesos ao longo da rede. Uma vez que sabemos o erro no output, nós usamos os pesos para trazê-lo de volta às camadas ocultas.

Perceptron Multicamadas - MPL

Cont… Backpropagation

Por exemplo, na camada de outuput, temos erros δ_k^o atribuídos para cada unidade de output k. Então, o erro atribuído para a unidade oculta j é igual aos erros dos outputs, proporcional aos pesos entre as camadas oculta e de output (levando o gradiente em conta):

$$\delta_j^h = \sum W_{jk} \delta_k^o f'(h_j)$$

Então, o passo do gradiente descendente é o mesmo que antes, apenas com os novos erros:

$$\Delta w_{ij} = \eta \delta_j^h x_i$$

onde w_{ij} são os pesos entre o inputs e a camada oculta e x_i são os valores de input da unidade. Esse formato continua válido para qualquer quantidade de camadas. O passos de peso são iguais ao tamanho do passo vezes o erro de output da camada vezes os valores de input daquela camada

$$\Delta w_{pq} = \eta \delta_{output} V_{in}$$

Aqui, temos o erro de output, δ_{output} , ao propagar os erros retrospectivamente de camadas mais altas. E os valores de input, V_{in} são os inputs da camada, as ativações da camada oculta para camada de output, por exemplo.

Perceptron Multicamadas - MPL

Exemplo no material auxiliar...

Implementação com Numpy

Você já possui a maior parte das coisas necessárias para implementar a backpropagation com o Numpy.

No entanto, antes só devíamos lidar com erros de uma unidade. Agora, na atualização dos pesos, temos que considerar o erro de *cada unidade* da camada oculta, δ_j: $\Delta w_{ij} = \eta \delta_j x_i$

Tarefa 5 - Backpropagation

Exercício de Backpropagation

Abaixo, você implementará o código para calcular uma rodada de atualização com backpropagation para dois conjuntos de pesos. Escrevi o andamento para frente, o seu objetivo é escrever o andamento para trás.

Coisas a fazer

- Calcular o erro da rede.
- Calcular o gradiente de erro da camada de output.
- Usar a backpropagation para calcular o erro da camada oculta.
- Calcular o passo de atualização dos pesos.