**[RNA.js](https://gist.github.com/CodeAkinori/b815e4b07c64e4199afbd257af27e44f" \l "file-rna-js)**

|  |  |
| --- | --- |
|  | // Função que gera um número aleatório entre 'min' e 'max' |
|  | function randomRange(min, max) { |
|  | return Math.random() \* (max - min) + min; |
|  | } |
|  |  |
|  | // Função de interpolação linear |
|  | // A interpolação linear é uma forma de adivinhar um valor entre dois números que você já conhece. Imagine esses dois números como pontos em uma linha. Se você quer encontrar um número entre esses dois pontos, a interpolação linear faz isso usando uma fórmula simples. Basicamente, ela "liga os pontos" de forma reta para pegar o valor intermediário. É muito útil quando precisamos de valore entre dois números conhecidos. |
|  |  |
|  | function lerp(a, b, t) { |
|  | // Calcula um valor intermediário entre 'a' e 'b' com base no fator 't' |
|  | return a + (b - a) \* t; |
|  | } |
|  |  |
|  | // Vamos criar a definição da classe Neuron (Neurônio) |
|  | class Neuron { |
|  | constructor(inputs) { |
|  | // Inicializa o neurônio com um viés (bias) aleatório no intervalo [-1, 1] |
|  |  |
|  | // O viés (bias) na nossa rede neural é um número que ajuda a rede a aprender. Ele permite que a rede faça ajustes para entender melhor os dados. O bias é importante porque ajuda a rede a aprender padrões nos dados e a tomar decisões melhores. É como um ajuste pequeno que a rede faz para se adaptar aos dados da melhor maneira possível. |
|  | this.bias = randomRange(-1, 1); |
|  |  |
|  | // Inicializa uma lista de pesos com valores aleatórios no intervalo [-1, 1] |
|  | this.weightList = new Array(inputs) |
|  | .fill() |
|  | .map(() => randomRange(-1, 1)); |
|  | }; |
|  |  |
|  | // Função que calcula a saída do neurônio (ativação) |
|  | g(signalList = []) { |
|  | let u = 0; |
|  |  |
|  | // Calcula a soma ponderada dos sinais de entrada multiplicados pelos pesos |
|  | for (let i = 0; i < this.weightList.length; i++) { |
|  | u += signalList[i] \* this.weightList[i]; |
|  | } |
|  |  |
|  | // Verifica se o neurônio está ativado com base na função tangente |
|  | if (Math.tanh(u) > this.bias) return 1; // Ativado |
|  | else return 0; // Não ativado |
|  | } |
|  |  |
|  | // Função que realiza mutação nos pesos e no viés do neurônio |
|  | mutate(rate = 1) { |
|  | this.weightList = this.weightList.map((w) => { |
|  | // Faz uma mudança nos pesos com base na taxa 'rate' |
|  | return lerp(w, randomRange(-1, 1), rate); |
|  | }); |
|  | // Faz uma mudança no viés (bias) com base na taxa 'rate' |
|  | this.bias = lerp(this.bias, randomRange(-1, 1), rate); |
|  | } |
|  | } |
|  |  |
|  | // Definição da classe RNA (Rede Neural Artificial) |
|  | class RNA { |
|  | constructor(inputCount = 1, levelList = []) { |
|  | // Inicializa a pontuação da RNA com zero |
|  | this.score = 0; |
|  |  |
|  | // Cria as camadas de neurônios com base nas especificações |
|  | this.levelList = levelList.map((l, i) => { |
|  | // Calcula o tamanho da camada atual |
|  | const inputSize = i === 0 ? inputCount : levelList[i - 1]; |
|  |  |
|  | // Cria uma camada de neurônios com o tamanho calculado |
|  | return new Array(l).fill().map(() => new Neuron(inputSize)); |
|  | }); |
|  | } |
|  |  |
|  |  |
|  |  |
|  | // Função que calcula a saída da RNA com base nos sinais de entrada |
|  | compute(list = []) { |
|  | for (let i = 0; i < this.levelList.length; i++) { |
|  | const tempList = []; |
|  | // Calcula a saída de cada neurônio na camada atual |
|  | for (const neuron of this.levelList[i]) { |
|  | if (list.length !== neuron.weightList.length) throw new Error('Entrada inválida'); |
|  | tempList.push(neuron.g(list)); |
|  | } |
|  | list = tempList; // Atualiza os sinais de entrada para a próxima camada |
|  | } |
|  | return list; // Retorna a saída final da RNA |
|  | } |
|  |  |
|  | // Função que realiza mutação em todos os neurônios da RNA |
|  | mutate(rate = 1) { |
|  | for (const level of this.levelList) { |
|  | for (const neuron of level) neuron.mutate(rate); |
|  | } |
|  | } |
|  |  |
|  | // Função para carregar a configuração de uma RNA existente |
|  | load(rna) { |
|  | if (!rna) return; |
|  | try { |
|  | this.levelList = rna.map((neuronList) => { |
|  | // O método .map() no JavaScript serve para criar uma nova lista usando a informação de uma lista original. |
|  | return neuronList.map((neuron) => { |
|  | // Cria novos neurônios com base nos dados da RNA carregada |
|  | const n = new Neuron(); |
|  | n.bias = neuron.bias; |
|  | n.weightList = neuron.weightList; |
|  | // O weightList no nosso caso, vai servir para atribuir importância a diferentes entradas, ajudando a nossa rede a aprender e tomar decisões com base nos dados que entram. |
|  | return n; |
|  | }); |
|  | }); |
|  | } catch (e) { |
|  | // `catch (e)` é vai ser usado para capturar e lidar com erros em JavaScript, permitindo que você tome decições específicas quando acontece uma exceção durante a execução do código. |
|  | return; |
|  | } |
|  | } |
|  |  |
|  | // Função para salvar a configuração atual da RNA |
|  | save() { |
|  | return this.levelList; |
|  | } |
|  | } |
|  |  |
|  | // Exporta a classe RNA como o valor padrão do módulo |
|  | export default RNA; |