RELATÓRIO SOBRE REDE NEURAL DE KOHONEN

CIC 260 - INTELIGÊNCIA ARTIFICIAL

Nome:	Willian Saymon da Silva
Matrícula:	33962
Linguagem:	Python 3

O ALGORITMO DE KOHONEN

Os mapas auto-organizáveis de Kohonen utilizam treinamento não supervisionado, onde a rede busca encontrar similaridades baseando-se apenas nos padrões de entrada. O principal objetivo dos mapas auto-organizáveis de Kohonen é agrupar os dados de entrada que são semelhantes entre si formando classes ou agrupamentos denominados clusters.

A entrada do problema consiste de um conjunto de vetores de características, estes vetores devem possuir o mesmo número de componentes para que o algoritmo funcione perfeitamente. Cada unidade de saída representa um grupo ou cluster, o que limita a quantidade de clusters ao número de saídas. Durante o treinamento a rede determina a unidade de saída que melhor responde ao vetor de entrada; o vetor de pesos para a unidade vencedora é ajustado de acordo com o algoritmo de treinamento a ser descrito na próxima seção.

Durante o processo de auto-organização do mapa, a unidade do cluster cujo vetor de pesos mais se aproxima do vetor dos padrões de entrada é escolhida como sendo a “vencedora”. A unidade vencedora e suas unidades vizinhas têm seus pesos atualizados segundo uma regra a ser descrita a seguir.

IMPLEMENTAÇÃO

Segue abaixo o algoritmo, orientado a objetos, para solução do problema:

```
1.  classe KohonenSOM:
2.
3.          Método  KohonenSOM(entradasP, pesosP):
4.                  entradas = entradasP
```

```
5.              num_máximo_grupos = 2
6.              fator_aprendizado = 0.6
7.              fator_aprendizado_mínimo = 0.000000001
8.
9.              pesos = pesosP
10.              num_pesos = tamanho(pesos)
11.
12.         Método inicia_matriz_pesos():
13.              pesos_aleatórios = []
14.              Para cada peso::
15.                   linha = []
16.                   Para cada grupo:
17.                        linha.adiciona(aleatório(0, 1))
18.                   pesos_aleatórios.adiciona(linha)
19.              pesos = pesos_aleatórios
20.
21.         Método treinar():
22.              interacoes = 0
23.              Loop:
24.                   Para cada  entrada :
25.                        menor_distancia = [0, 99999999]
26.                        Para cada grupo:
27.                             distância = calcula_distância(grupo, entrada)
28.                             Se distancia menor que menor_distância[1]:
29.                                  menor_distância = [grupo, distância]
30.
31.                        atualiza_peso_grupo(menor_distância[0], entrada)
32.                   atualiza_fator_aprendizado()
33.
34.              Se fator_aprendizado menor ou igual ao fator_aprendizado_mínimo:
35.                   Parar
36.              interacoes += 1
37.
38.         Método calcula_distância(grupo,entrada):
39.              distancia = 0
40.              Para cada peso:
41.                   distância += (pesos[peso][grupo] - entradas[entrada][peso]) ^ 2
42.              retorna distância
43.
44.         Método atualiza_peso_grupo( grupo, entrada):
45.              Para cada peso:
46.                   pesos[peso][grupo] += fator_aprendizado *
                                  (entradas[entrada][peso] -.pesos[peso][grupo])
47.                   pesos[peso][grupo] = round(pesos[peso][grupo], 5)
48.
49.         Método  atualiza_fator_aprendizado():
50.              fator_aprendizado = fator_aprendizado * 0.5
51.
52.         Método adicionar_entrada( entrada):
```

```
53.             num_pesos = tamanho(entrada)
54.             entradas.adiciona(entrada)
```

O algoritmo apresentado segue a seguinte lógica de execução:

1. Fator de correção e demais variáveis. Entradas e pesos devem passados por parâmetro!
2. Para cada uma das entradas;
    2.1. Calcula a distância/semelhança com os grupos;
    2.2. Seleciona o grupo com a qual mais se assemelha;
    2.3. Adiciona ao grupo suas características baseado no fator de aprendizagem;
3. Atualiza o fator de aprendizagem;
4. Verifica se o fator de aprendizagem é maior que o mínimo;
    4.1. Se sim, retorna a etapa 2;
    4.2. Se não, encerra o treinamento;

BIBLIOGRAFIA:

COPPIN, Ben. **Artificial intelligence illuminated**. Jones & Bartlett Learning, 2004.

FAUSETT, Laurene; FAUSETT, Laurene. **Fundamentals of neural networks: architectures, algorithms, and applications**. Prentice-Hall, 1994.