Implementação de Mapas Auto-Organizáveis de Kohonen para reconhecimento de caracteres

Elias Fank[1], **João Gehlen**[1], **Ricardo Zanuzzo**[1]

[1]Curso de Ciência da Computação – Universidade Federal da Fronteira Sul (UFFS)
Caixa Postal 181 – 89.812-000 – Chapecó – SC – Brasil

eliasfank@hotmail.com, joaogehlen91@gmail.com, zanuzzorz@gmail.com

Resumo. *Este documento tem o objetivo de descrever o funcionamento dos mapas auto-organizáveis de Kohonen e explicar a implementação do algoritmo cujo objetivo é,reconhecer números escritos à mão que estão representados em um arquivo texto, que será utilizado para fazer o treinamento da rede e posteriormente a classificação, com o objetivo de analisar qual a precisão do algoritmo. Além disso, mostrar os resultados obtidos.*

1. Introdução

Um algoritmo de Rede Neural tem o objetivo de reconhecer, treinar um conjunto de neurônios e então classificar valores da entrada utilizando os neurônios treinados. Um bom algoritmo para fazer o reconhecimento de caracteres é o Mapa Auto-Organizável de Kohonen. Este trabalho apresenta uma implementação do mapa de Kohonen e os resultados obtidos através deste método.

Na primeira parte é feito uma contextualização sobre os Mapas Auto-Organizáveis de Kohonen, em seguida é descrito como foi feito a codificação, depois é apresentado os resultados e por fim a conclusão.

2. Mapas Auto-Organizáveis de Kohonen

Self-Organization Maps(SOM), ou Mapas Auto-Organizáveis de Kohonen fazem parte de um grupo de redes neurais chamado de redes competitivas. Este tipo de rede, utiliza competição junto com uma forma de aprendizagem para ir atualizando os seus pesos de cada elemento da rede.[Von Zuben 2005]

Este tipo de rede, utiliza um treinamento não-supervisionado, que significa que a rede busca encontrar similaridade com base apenas nos valores de entrada. Os mapas auto-organizáveis de Kohonen tem o agrupar os dados da entrada que são semelhantes entre si, formando grupos chamados de *clusters*.[Von Zuben 2005]

A rede classificadora é formada por um conjunto de neurônios, esses neurônios depois de treinados ficam agrupados em grupos de neurônios mais parecidos entre si. Durante o treinamento a rede determina o neurônio que melhor representa o valor da entrada. O vetor de pesos do neurônio vencedor é atualizado, assim como os vizinhos são também atualizados, seguindo uma regra que será explicada ao decorrer do texto.

2.1. Arquiteturas de um Mapa Auto-Organizável de Kohonen

Existem duas arquiteturas mais utilizadas em mapas de Kohonen, que são: unidimensional e bidimensional, conforme ilustrado na figura 1. Na arquitetura bidimensional pode se dizer que os neurônios estão organizados em forma de uma matriz, enquanto que na unidimensional os neurônios estão organizados em um vetor [Kohonen 1990, Kohonen 1998].

A quantidade de neurônios pode ser mantida fixa, definida pelo usuário na forma de dimensão da matriz, ou então a quantidade para a arquitetura unidimensional. A vantagem de usar uma arquitetura unidimensional é quando a quantidade de neurônios é gerada automaticamente pelo algoritmo.

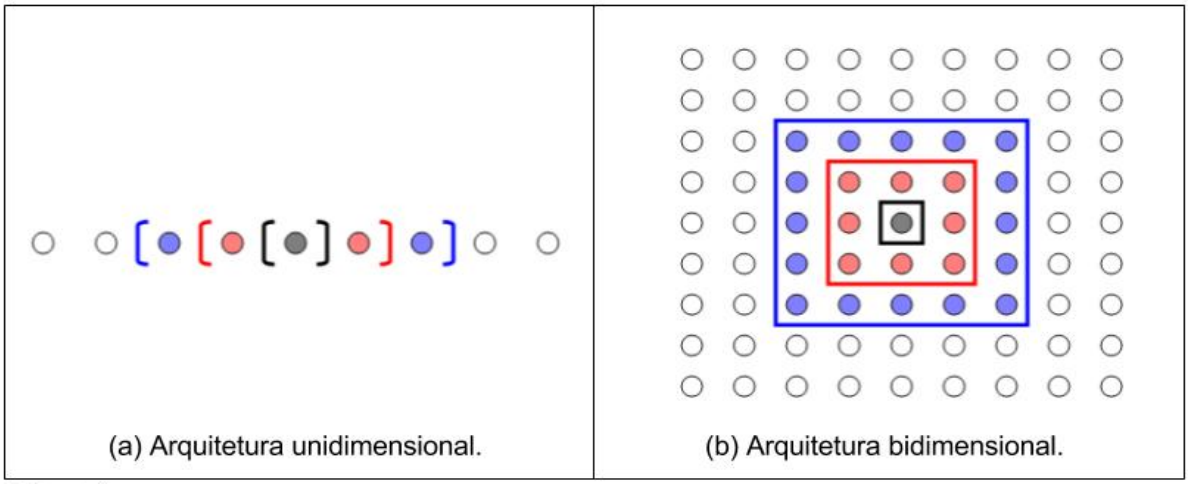

Figure 1. Tipos de arquiteturas mais usadas em uma rede de Kohonen

3. Codificação

A estrutura do programa foi dividido em quatro etapas:

- Geração da Matriz aleatória de pesos: Nesta etapa, o programa gera uma matriz de neurônios aleatória, variando os pesos entre 0 e 1.
- Leitura do arquivo: Nesta etapa, o programa faz a leitura do arquivo de entrada, onde é feita a separação do arquivo em duas partes, conjunto de treinamento e conjunto de teste. O conjunto de treinamento, convencionalmente representando 70% do arquivo de entrada, é utilizado para treinar a rede. Já o conjunto de teste, formado pelos 30% restantes, é utilizado para fazer a classificação e apurar a acurácia da rede.
- Treinamento: Nesta etapa, o programa itera sobre cada valor da entrada do conjunto de treinamento, encontrando o neurônio que melhor representa esta entrada, definindo-o como vencedor. Feito isso, deve-se atualizar os pesos de todos os neurônios(conforme mostrado na figura 2) levando em consideração a posição deles com relação ao neurônio vencedor. Este processo de treinamento se repete para cada época. A função
- Teste: Nesta etapa, o programa itera sobre cada valor do conjunto de teste, encontrando o neurônio que mais representa este valor. Depois de encontrar o valor, o

programa compara a *label* do neurônio com a *label* do valor da entrada do conjunto de teste. Se caso os dois valores são iguais, então soma-se 1 na quantidade de acertos, caso contrário, soma-se 1 na quantidade de erros. Então com essa informações de acertos e erros, o programa calcula a acurácia de cada *cluster* e também a acurácia da rede. Este processo é feito na função apresentada na Figura 3

```
public void treinamento(String[] label){
  double[] bmu = new double[1024];
  double raio, taxaAp;

  for (int epoca = 1; epoca <= this.maxEpocas ; epoca++) {
    raio = raio_vizinhanca(epoca);
    System.out.println("Epoca:"+ epoca);
    System.out.println("Raio:"+ raio);
    taxaAp = this.taxa_aprendizado(epoca);

    for (int x = 0; x < trainSet.length; x++) {
      maisProx(trainSet[x]);
      bmu = mapa[this.melhorI][this.melhorJ];
      mapaRotuladoTrain[this.melhorI][this.melhorJ] = label[x];

      atualiza_pesos(x, bmu, taxaAp, raio);
    }
  }
}

public void atualiza_pesos(int x, double[] bmu, double taxaAp, double raio){
  double delta, influencia_vizinhanca;
  for(int i = 0; i< this.dimMatNeuronios; i++){
    for(int j = 0; j< this.dimMatNeuronios; j++){
      for(int k = 0; k< 1024; k++){
        influencia_vizinhanca = this.influencia_vizinhanca(bmu, this.melhorI, this.melhorJ, i, j, raio);
        delta = taxaAp * influencia_vizinhanca * (trainSet[x][k] - mapa[i][j][k]);
        mapa[i][j][k] += delta;
      }
    }
  }
}
```

Figure 2. Funções que fazem o controle do treinamento e atualização dos pesos

4. Resultados

Nesta seção é apresentado um exemplo de execução e os resultado obtidos através deste exemplo. Foi utilizado os seguintes parâmetros:

- dim_neuronio = 15: Matriz de neurônios de tamanho 15 x 15.
- max_epocas = 20: Representa o número de treinamentos.
- taxa_ap = 0.1: Representa a taxa inicial de aprendizagem.
- raio = 0.5: Representa o tamanho do raio inicial usado no calculo da vizinhança.
- test_size = 0.3: Representa o percentual dos valores de entrada que serão usados para o teste.

Com esse parâmetros foi obtido os resultados mostrados na figura 4 e na tabela 1.

```
public void teste(int test_set[][], String labels[]){
  for(int i=0;i<test_set.length;i++){
    maisProx(test_set[i]);
    mapaRotulado[this.melhorI][this.melhorJ] = labels[i];
    int n = Integer.parseInt(labels[i]);
    if(labels[i].equals(mapaRotuladoTrain[this.melhorI][this.melhorJ])){
      this.acuracia[0][n] += 1;
    }else{
      this.acuracia[1][n] += 1;
    }
  }
}
```

Figure 3. Função que faz a classificação dos valores de conjunto de teste

```
##################################
## 3 3 3 3 3 5 9 9 9 9 . . 0 0 0 ##
## 3 3 3 3 . 9 9 9 . 9 9 0 0 0 0 ##
## 3 3 3 3 9 9 . 5 5 9 0 0 0 0 0 ##
## 3 . 2 . 9 . 5 5 5 5 . 0 0 . 0 ##
## . . 2 2 2 . 5 5 5 . 8 8 6 . 6 ##
## 2 2 2 2 . 5 5 5 5 . 8 8 5 6 6 ##
## 2 2 . 1 . 5 5 5 3 8 8 8 6 6 6 ##
## 2 2 . 1 1 5 . . 8 8 8 8 6 . 6 ##
## 2 . . 8 5 5 . 8 8 8 8 8 6 6 6 ##
## 2 2 7 8 8 1 1 . 1 1 . 8 6 . 6 ##
## . . . 8 . . 1 1 1 1 1 . 6 6 6 ##
## 7 7 7 9 4 4 . 1 . 1 1 . 4 4 4 ##
## 7 7 7 . . 4 . . 1 1 1 4 4 4 4 ##
## 7 7 . 7 . 9 5 . 9 1 1 4 4 4 4 ##
## 7 7 7 7 7 9 9 9 1 1 7 4 4 4 4 ##
##################################
```

Figure 4. Exemplo do arquivo de saída.

Cluster	0	1	2	3	4	5	6	7	8	9	TOTAL
% acerto	98.43	92.72	96.61	92.30	74.54	89.47	96.61	97.01	91.07	82.14	91.37

Table 1. Acurácia de cada *cluster*

5. Conclusão

Ao fim do trabalho, pode-se concluir que os objetivos foram atingidos. O presente trabalho teve como objetivo desenvolver um reconhecedor de caracteres usando redes neurais (usou-se o algoritmo *Kohonen*). No trabalho, o algoritmo de *Kohonen* foi implementado sem a utilização de bibliotecas. Isso permitiu perceber que o algoritmo é simples de ser implementado. Contudo, a complexidade computacional do algoritmo é bastante alta devido a dimensão dos dados de entrada.

References

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480.

Kohonen, T. (1998). The self-organizing map. *Neurocomputing*, 21(1):1–6.

Von Zuben, F. (2005). Rede neural de kohonen e aprendizado não-supervisionado. *Notas de Aula do Curso IA353, 1º semestre de*.