Rede neural para reconhecimento de caracteres

Acácia dos Campos da Terra, João Pedro Winckler Bernardi

[1]Universidade Federal da Fronteira Sul (UFFS)
Caixa Postal 181 – 89802-112 – Chapecó – SC – Brasil

terra.acacia@gmail.com, jpwbernardi@hotmail.com

Resumo. *Este documento tem por função descrever o programa cujo objetivo é, dado um caracter escrito a mão, determinar qual é esse carácter. Esse procedimento é realizado através de uma rede neural, que foi treinada para tal. Neste trabalho estarão descritos os resultados obtidos.*

Introdução

O algoritmo de Kohonen é relativamente simples e possui a capacidade de organizar dados em grupos. Então, utilizando uma matriz de neurônios como rede neural e treinando a rede através do algoritmo de Kohonen, conseguimos organizar em grupos os dados. Neste trabalho, os dados são caracteres numéricos. Assim, a rede se auto-organizará, separando em grupos as características de cada número. Para reconhecer um carácter, verifica-se qual neurônio o reconhece e em qual grupo este neurônio esta. Isso será melhor explicado quando for falado sobre os resultados.

Algoritmo de Kohonen

O algoritmo de Kohonen, que consiste no treinamento da rede neural, está descrito a seguir.

Algoritmo 1: KOHONEN

Entrada: $d : data$

1 Randomize os valores dos neurônios da rede
2 **para** *cada neurônio no mapa* **faça**
3 Use a distância euclidiana para encontrar o neurônio mais similar com a entrada, ou seja, o com menor distância(BMU).
4 **fim**
5 Atualize os neurônios na vizinhança do BMU, aproximando-os da entrada.
6 Repita o processo enquanto houverem entradas ou a quantidade de iterações desejada não foi atingida.

Implementação

A rede de neurônios é representada por uma matriz de neurônios. Cada posição da matriz consiste num neurônio que é representado por um vetor de *double*. Ao início do processo, todo neurônio é preenchido com valores aleatórios de 0 a 1. Então, a partir da entrada de treinamento obtida a partir da base de dados da UCI, a rede é treinada. Todas as entradas são guardadas num vetor, que é embaralhado. Seja D o vetor de dados e N a matriz de neurônios, ao descobrir o BMU, a vizinhança é atualizada com a fórmula:

$$N[i][j] = (D - N[i][j]) * LEN/(8.0 * distBMU(iBMU, jBMU, i, j)) * \alpha;$$

- i e j: representam a posição do neurônio que está sendo atualizado.
- $iBMU$ e $jBMU$: representam a posição do BMU.
- $distBMU$: é a função que calcula a distância do neurônio N[i][j] do BMU na rede.
- LEN: é a quantidade de linhas/colunas da entrada, neste trabalho é 32.
- $LEN/(8.0 * distBMU(iBMU, jBMU, i, j)$: representa a restrição devido a distancia do BMU.
- α: é a restrição de aprendizagem. Normalmente, considera-se 0,1.

Esse processo todo pode ser visto no código nesses trechos:

```
void leTreino() {
...
//Itera-se pela quantidade de vezes que
//será realizado o treinamento
  for (j = 0; j < QTDTREINO; j++) {
    //Randomiza-se a entrada
    random_shuffle(entrada.begin(), entrada.end());
    //Para cada caracter da entrada, treina-se a rede
    for (i = 0; i < (int)entrada.size(); i++)
      treina(entrada[i].first);
  }
...
}

void treina(vector<double> in) {
  //Descobre-se o BMU
  pair<int, int> _bmu = bmu(in);
  //Atualiza a vizinhança
  attVizinhanca(in, _bmu.first, _bmu.second);
}

void attVizinhanca(vector<double> in, int iBMU, int jBMU) {
    for (int i = 0; i < NEURONIOS; i++)
        for (int j = 0; j < NEURONIOS; j++) {
            double _const = LEN /
                        (8.0 * distBMU(iBMU, jBMU, i, j)) * ALPHA;
            for (int k = 0; k < LEN * LEN; k++)
                n[i][j][k] += ((in[k] - n[i][j][k]) * _const);
        }
}
```

É importante ressaltar que as entradas estão no formato de uma matriz de tamanho 32 por 32, porém, para facilitar o processo, todas as linhas são concatenadas num vetor. Por isso, os neurônios no trabalho também são vetores.

Treinamento

Resultados

References