Rede neural para reconhecimento de caracteres numéricos

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 trabalho feito para a disciplina de Inteligência Artificial da UFFS (Universidade Federal da Fronteira Sul), onde foi desenvolvido um programa com o objetivo de, dado um caractere escrito a mão, determinar qual é esse caractere. Para tal, é necessária a criação e treinamento de uma rede neural. Ao final deste relatório, serão descritos os resultados obtidos com o programa.*

1. Introdução

O algoritmo de Kohonen foi desenvolvido por Teuvo Kohonen em 1982 e por isso leva este nome. Utiliza o método de aprendizagem por competição (*competitive learning*), uma forma de aprendizagem não supervisionada em redes neurais artificiais, em que os nós competem pelo direito de responder a um subconjunto dos dados de entrada [Wikipedia 2017a]. O algoritmo possui a capacidade de organizar dimensionalmente dados complexos em grupos, também chamados de *clusters*, de acordo com suas relações [Wikipedia 2017b].

Neste trabalho, foi utilizada uma matriz de neurônios como rede neural e o treinamento da rede foi feito através do algoritmo de Kohonen. Dessa forma, foi possível organizar os dados em grupos. Neste caso, os dados são caracteres numéricos de zero à nove.

O algoritmo faz com que a rede se auto-organize, separando em grupos as características de cada número, de forma que serão gerados então pelo menos 10 grupos (um para cada número). Pode acontecer de o algoritmo gerar mais de 10 grupos, pois pode gerar dois grupos diferentes que reconhecem o mesmo número.

Para reconhecer um caractere, é verificado qual neurônio o reconhece e então em qual grupo este neurônio está. Assim é identificado qual o número reconhecido. Ao final deste relatório há uma sessão para os resultados obtidos, onde mais detalhes serão dados.

2. Algoritmo de Kohonen

O algoritmo de Kohonen, utilizado no trabalho para o treinamento da rede neural gera grupos, também chamados de *clusters*. Estes grupos servem para identificar determinadas características. Neste trabalho, os *clusters* identificam caracteres numéricos, sendo que cada *cluster* tem por função identificar um número distinto. Apesar disso, dois *clusters* podem identificar um mesmo número.

O algoritmo 1 abaixo descrito é responsável pelo treinamento da rede e criação dos *clusters*.

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.

3. Implementação

A rede de neurônios é representada no código por uma matriz de neurônios. Cada posição desta matriz consiste num neurônio que é representado por um vetor de *double*. Ao início do processo, todos os neurônios são preenchidos com valores aleatórios entre 0 e 1. Então, a partir de uma entrada de treinamento obtida a partir da base de dados da UCI *Machine Learning Repository*, a rede é treinada. Todas as entradas são guardadas num vetor, o qual é embaralhado.

Seja D o vetor de dados e N a matriz de neurônios, ao descobrir o BMU (*Best Matching Unit*), ou seja, o neurônio mais parecido com a entrada, a vizinhança é atualizada com a fórmula:

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

Onde:

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

O processo descrito pode ser visualizado em código nos trechos que seguem:

```
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 também são vetores.

4. Treinamento

Para o treinamento, foi utilizada uma entrada de treinamento disponibilizada pela base de dados da UCI *Machine Learning Repository*, conforme solicitado na descrição do trabalho. Para cada iteração de treinamento, foi feita a aleatorização da ordem da entrada de treinamento, e a partir disso a execução do algoritmo descrito anteriormente.

A seguir são exibidas tabelas que mostram como ficou a rede neural após: mil execuções, três mil execuções e cinco mil execuções. Cada treinamento demora um intervalo de tempo entre 4 e 8 horas.

Cada célula nas tabelas corresponde a um neurônio do programa e os valores dentro das células correspondem ao número que foi identificado por aquele neurônio. Para construção da tabela de identificação, foi usado o arquivo de testes também fornecido pela base de dados da UCI *Machine Learning Repository*.

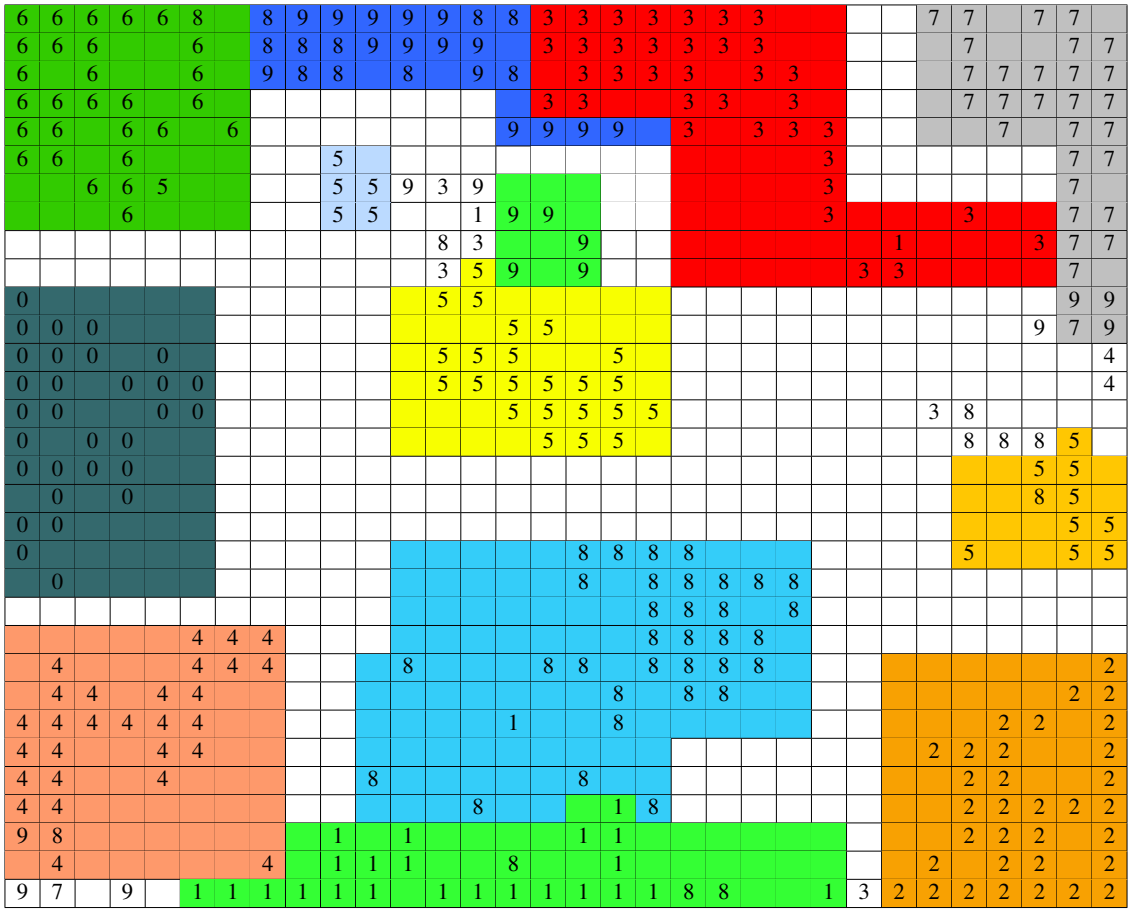

Tabela 1. Rede após o treinamento ter sido executado mil vezes

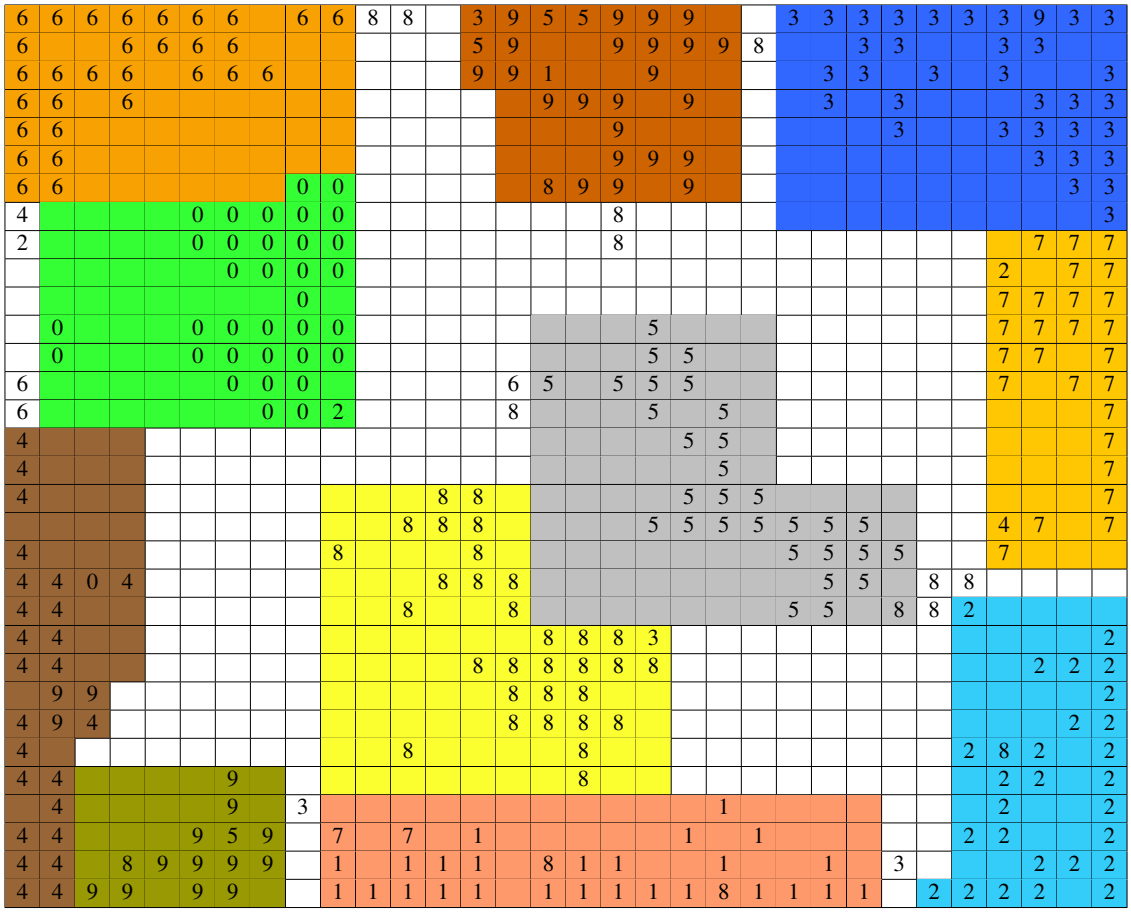

Tabela 2. Rede após o treinamento ter sido executado três mil vezes

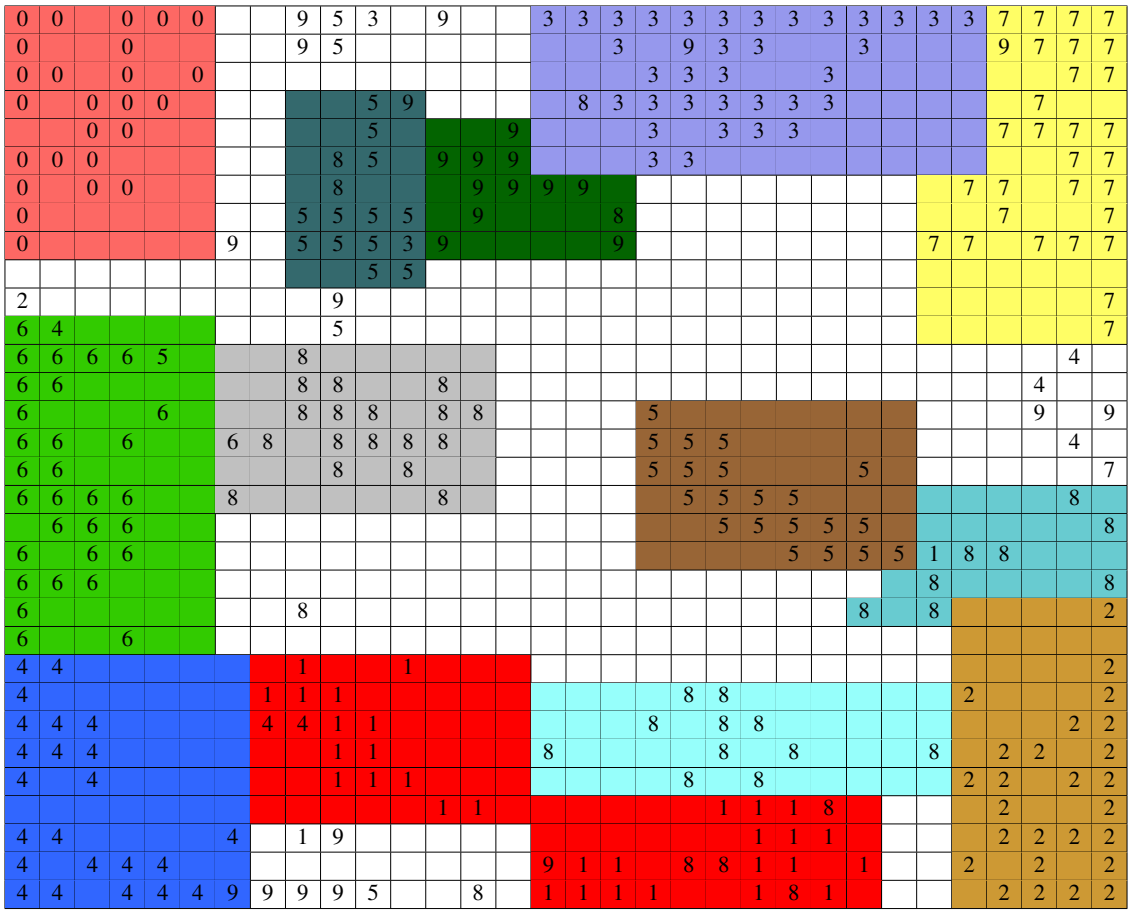

Tabela 3. Rede após o treinamento ter sido executado cinco mil vezes

Destas três redes neurais (Tabela 1, Tabela 2 e Tabela 3), pode-se observar que as mais consistente são as duas primeiras. Talvez se o treinamento fosse executado mais vezes, eventualmente a rede convergiria para uma rede melhor. É possível observar também que nas três tabelas, pelo menos um número está dividido em dois *clusters*, o que não é muito desejável.

Para a execução de um último teste, foi levemente alterada a função de restrição devido a distancia do BMU. A nova fórmula está abaixo exibida, onde pode-se observar que apenas a constante foi alterada.

$$N[i][j] = N[i][j] + [(D - N[i][j]) * LEN/(16.0 * distBMU) * \alpha];$$

O resultado obtido com o programa para um treinamento de duas mil iterações está exibido na Tabela 4.

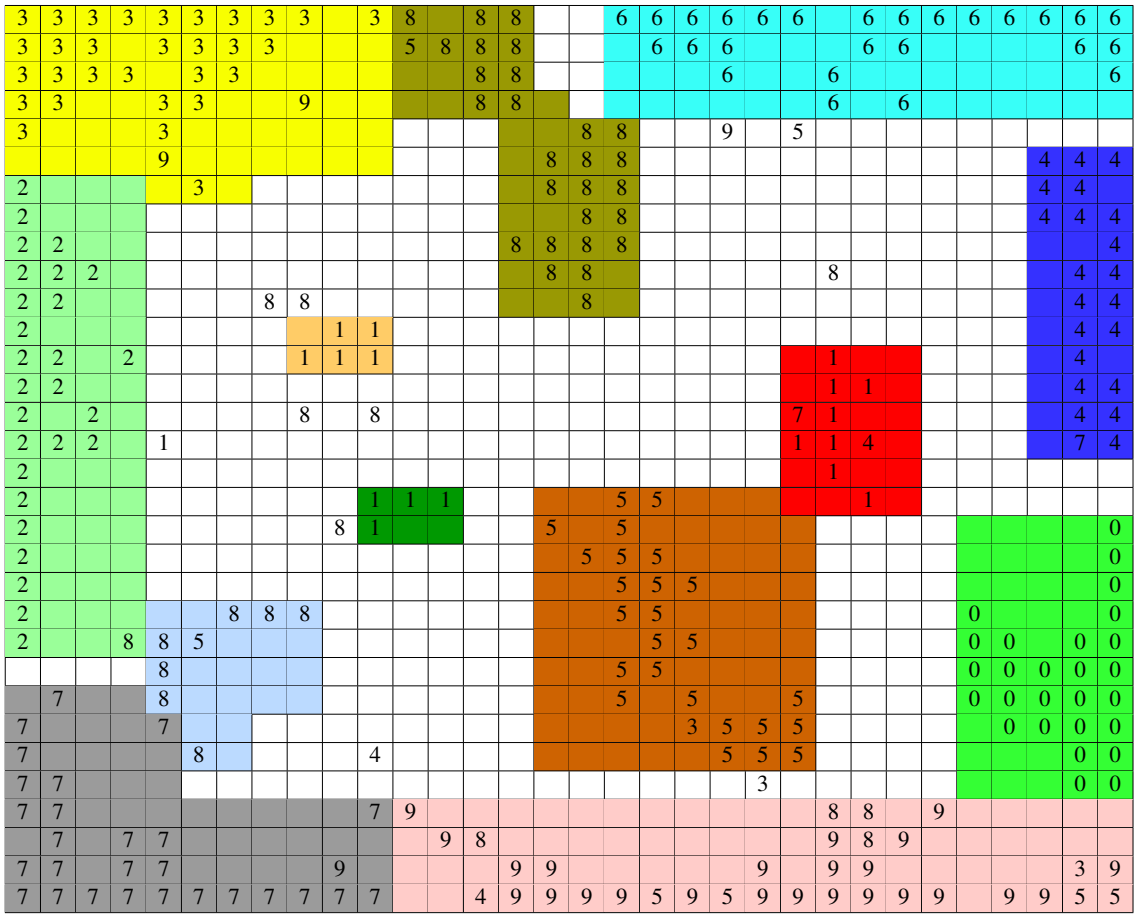

Tabela 4. Tabela com outra função de restrição devido a distancia do neurônio ao BMU com duas mil iterações do treinamento

Ao efetuar a comparação entre as tabelas, é possível perceber que a função de restrição devido a distância de um neurônio ao BMU é muito importante para a qualidade da rede gerada. Para os resultados apresentados, a rede escolhida foi a da Tabela 2, pois é a que parece mais consistente.

5. Resultados

Com os resultados obtidos do programa, são feitas duas comparações. Primeiramente, será considerada a Tabela 2 como rede. O número que se encontra na célula da tabela é o número que aquele neurônio identificará.

Executando a identificação do arquivo de testes do banco de dados da UCI com 943 entradas, foi obtida uma taxa de acerto de 97,9%, apresentada na Tabela 5.

Cada uma das colunas da Tabela 5 representa um dos caracteres reconhecidos, de 0 à 9, e contém informações de: Quantidade de entradas, quantidade de acertos e porcentagem de acerto. A última coluna destina-se a avaliar o desempenho geral, apresentando as mesmas informações, porém para todos os caracteres.

Caractere	0	1	2	3	4	5	6	7	8	9	TOTAL
Quantidade de entradas	100	94	93	105	87	81	95	90	109	89	943
Acertos	100	92	92	104	85	81	95	87	102	86	924
Porcentagem de acerto	100%	97,8%	98,9%	99%	97,7%	100%	100%	96,6%	93,5%	96,6%	97,9%

Tabela 5. Análise do resultado obtido para cada caractere

Pode-se observar, pela mesma tabela, que o caractere com menor taxa de acerto é o 8, sendo a taxa 93,5%. De fato, em muitas situações, 8 é um número que pode ser facilmente confundido com os demais números (como por exemplo o 3, que pode ser visto como metade de um 8). Existem situações também onde, à partir do 8, são gerados os demais números. É o caso dos números feitos com LEDs de 7 segmentos, como o visto na Figura 1.

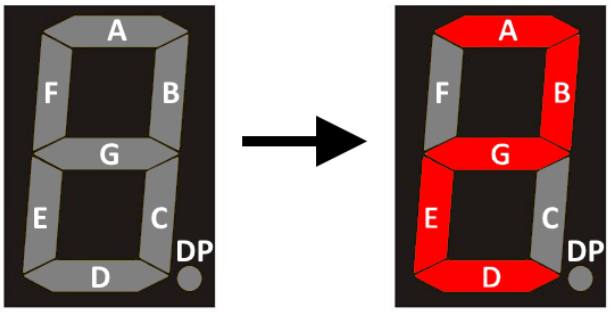

Figura 1. Exemplo de *display* de 7 segmentos [Filipeflop 2017]

Os demais números obtiveram uma porcentagem acima de 96% de acerto.

As falhas geradas pelo programa ao utilizar a Tabela 2 como rede neural estão apresentadas na Tabela 6. Nesta, a primeira coluna representa o caractere que a rede deveria ter reconhecido, a segunda coluna contém qual caractere a rede reconheceu e a terceira coluna quantas vezes esse mesmo erro se repetiu.

Caractere	Reconhecido	Quantidade de enganos
1	8	1
	9	1
2	1	1
3	1	1
4	1	1
	6	1
7	4	1
	9	2
8	1	4
	2	3
9	7	3

Tabela 6. Lista de erros da rede, considerando a Tabela 2 como rede neural.

Referências

Filipeflop, B. (2017). Como construir um relógio com arduino. Disponível em: http://blog.filipeflop.com/arduino/como-construir-um-relogio-com-arduino.html. Acesso em 15/07/2017.

Wikipedia (2017a). *Competitive Learning*. Disponível em: https://en.wikipedia.org/wiki/Competitive_learning. Acesso em 15/07/2017.

Wikipedia (2017b). Mapas de Kohonen. Disponível em: https://pt.wikipedia.org/wiki/Mapas_de_Kohonen. Acesso em 15/07/2017.