Reconhecedor de Caracteres Utilizando Rede Neural de Kohonen

Gabriel Batista Galli[1], **Matheus Antonio Venancio Dall'Rosa**[1], **Vladimir Belinski**[1]

[1]Ciência da Computação – Universidade Federal da Fronteira Sul (UFFS)
Caixa Postal 181 – 89.802-112 – Chapecó – SC – Brasil

{g7.galli96, matheusdallrosa94, vlbelinski}@gmail.com

Resumo. *O presente trabalho, apresentado ao curso de Ciência da Computação da Universidade Federal da Fronteira Sul - UFFS - Campus Chapecó - como requisito parcial para aprovação no Componente Curricular Inteligência Artificial, 2017.1, sob orientação do professor José Carlos Bins Filho, consiste na descrição da implementação de um reconhecedor de caracteres utilizando Rede Neural de Kohonen, bem como dos resultados obtidos.*

1. Introdução

Mapas Auto-Organizáveis (*Self-Organizing Maps (SOMs)*, em inglês) são redes neurais não supervisionadas baseadas em aprendizado competitivo, uma forma de regra de aprendizado onde os neurônios competem entre si para ser decidido qual deles será ativado e, desta forma, tem-se a ativação de somente um neurônio a partir da entrada de um padrão [Filho 2017].

Redes Neurais de Kohonen são um tipo de SOM [Filho 2017] cujo algoritmo apresenta a capacidade de organizar dados em grupos. Sabendo-se disso, neste trabalho será feito uso deste tipo de rede para agrupamento/reconhecimento de caracteres numéricos. Na Seção 2 será provida um descrição acerca da preparação dos dados e da rede utilizada. Por sua vez, na Seção 3 serão apresentados os resultados alcançados através da implementação realizada.

2. Preparação dos dados e descrição da rede utilizada

Os dados utilizados para treinamento e testes da rede neural implementada foram obtidos da base de dados da *University of California, Irvine (UCI)*, localizada em `https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits`. No trabalho, os dados para treinamento e testes podem ser visualizados nos arquivos *optdigits-orig.tra* e *optdigits-orig.tes*, respectivamente. Cabe destacar que nestes arquivos cada caractere numérico é representado por uma matriz de 32×32 elementos, onde cada elemento consiste em um dígito 0 ou 1 e o conjunto de $1s$ representa o traçado do caractere numérico.

O algoritmo de treinamento da Rede Neural de Kohonen implementada neste trabalho foi baseado nos materiais de [Filho 2017] e [Wikipedia 2017], sendo que seu pseudo-código, bem como a inicialização dos neurônios e decremento dos parâmetros α (taxa de aprendizado) e σ (parâmetro de restrição da vizinhança) é realizado tal como apresentado no Algoritmo 1. Por sua vez, cada iteração do treinamento consiste nos passos apresentados no Algoritmo 2.

A implementação do treinamento da rede neural deste trabalho pode ser encontrada na função `train_neurons()` do arquivo `main.cpp`.

Antes da apresentação do Algoritmo 1 cabe definir a estrutura de um neurônio. Na implementação realizada, cada neurônio é formado por uma matriz de 32×32 números de ponto flutuante que representam seu peso, sendo esses números inicializados com valores contidos no intervalo real $[0, 1]$. Além disso, cada neurônio também possui um vetor de 10 posições, nomeado `digits`, para fins de contagem da frequência de reconhecimento de cada dígito. A utilidade desse vetor será melhor apresentada na seção 3.

Algoritmo 1: Algoritmo de treinamento.

Entrada: T: vetor de dígitos de treino.
Seja α a taxa de aprendizado, inicializada com 0.1.
Seja σ o parâmetro de restrição da vizinhança, inicializado com 2.
Seja M um mapa de neurônios de dimensões 24×24 e com pesos aleatórios contidos no intervalo real $[0, 1]$.
para *i de* 0 *até* 999 **faça**
    Utilize o Algoritmo 2 para treinar o mapa M.
    Decremente α em 0.00009 e σ em 0.0015.
fim

Antes da apresentação do Algoritmo 2 cabe definir os cálculos de distância euclidiana quadrada utilizados por ele.

Definição 2.1. *Distância euclidiana quadrada entre duas matrizes.* Sejam A e B duas matrizes $n \times n$, a distância euclidiana quadrada entre A e B é dada por:

$$dist1(\mathbf{A}, \mathbf{B}) = \sum_{i=1}^{n} \sum_{j=1}^{n} (a_{ij} - b_{ij})^2 \tag{1}$$

Na implementação realizada esse cálculo é encontrado na função `sq_euclidean_distance(Matrix &a, Matrix &b)` de `main.cpp`.

Definição 2.2. *Distância euclidiana quadrada entre dois neurônios/pontos.* Seja um neurônio na posição (m, n) do mapa de neurônios M, a distância euclidiana quadrada desse neurônio em relação a outro neurônio na posição (x, y) é dada por:

$$(m - x)^2 + (n - y)^2 \tag{2}$$

Na implementação realizada esse cálculo é encontrado na função

`sq_euclidean_distance(ii a, ii b)` de `main.cpp`.

Algoritmo 2: Iteração do algoritmo de KOHONEN

Entrada: T: vetor de dígitos de treino.

1 Sejam α, σ e M definidos tal como no Algoritmo 1.
2 Seja D o conjunto de dígitos de T em uma ordem aleatória.
3 Seja w um neurônio de M e x e y suas coordenadas.
4 Seja $dist1(A, B)$ a distância euclidiana quadrada tal como exposto na Definição 2.1.
5 Seja $dist2(a, b)$ a distância euclidiana quadrada tal como exposto na Definição 2.2.
6 **para** *cada dígito D_i (dígito i de D)* **faça**
7 Escolha o neurônio mais próximo de D_i (BMU) utilizando a função de distância $dist1()$.
8 Atualize cada neurônio $w_{x,y}$ de M de acordo com a fórmula:

$$w_{x,y} = w_{x,y} + \alpha \times e^{\frac{-dist2(BMU, w_{x,y})}{\sigma}} \times (D_i - w_{x,y})$$

9 **fim**

Através dos pseudo-códigos apresentados em Algoritmo 1 e Algoritmo 2 é notável que na implementação realizada são executadas 1000 iterações para treinamento, utilizado um mapa de 24×24 neurônios e que a taxa de aprendizagem (α) inicial (0.1) é decrementada a cada iteração, mas finaliza em valor maior que 0.

Além disso, em relação ao cálculo da vizinhança, tem-se que esse é realizado através de uma travessia completa do mapa de neurônios, sendo a atualização dos seus pesos limitada pela função exponencial de Euler, na qual é utilizado o parâmetro σ, relacionado à restrição da vizinhança. Cabe destacar que (σ) é inicializado com o valor 2 e decrementado ao longo das iterações, de forma com que a vizinhança de um neurônio se torne cada vez mais restrita conforme for ocorrendo a etapa de treinamento. Ademais, tal como α, σ também finaliza com valor maior que 0.

3. Montagem dos *clusters*, testes e resultados alcançados

Nesta seção será apresentado como foi realizada a montagem dos *clusters*, quais foram os testes realizados, bem como apresentados os resultados obtidos.

Em relação à montagem dos *clusters*, implementada em `match_training()`, tem-se que para realizá-la o arquivo de treinamento (*optdigits-orig.tra*) é processado novamente, mas sem alteração dos neurônios. Para cada dígito D_i de treino é encontrado um neurônio n_j, tal que a distância de D_i para n_j é menor do que a distância de D_i para qualquer outro neurônio. Feito isso, incrementa-se o número de ocorrências de D_i no vetor `digits` de n_j. Assim, o dígito de maior ocorrência em um neurônio define o caractere que o neurônio representa. Desta maneira definem-se os *clusters* como grupos de neurônios que representam um mesmo dígito.

Por sua vez, nos testes (implementados em `run_test()`), para cada dígito D_u de teste é encontrado um neurônio n_v, tal que a distância de D_u para n_v é menor do que a distância de D_u para qualquer outro neurônio. A partir disso, toma-se o dígito representado por n_v como o caractere reconhecido pela rede para D_u. Finalmente, conta-se quantos dos caracteres reconhecidos correspondem ao dígito que D_u de fato representa.

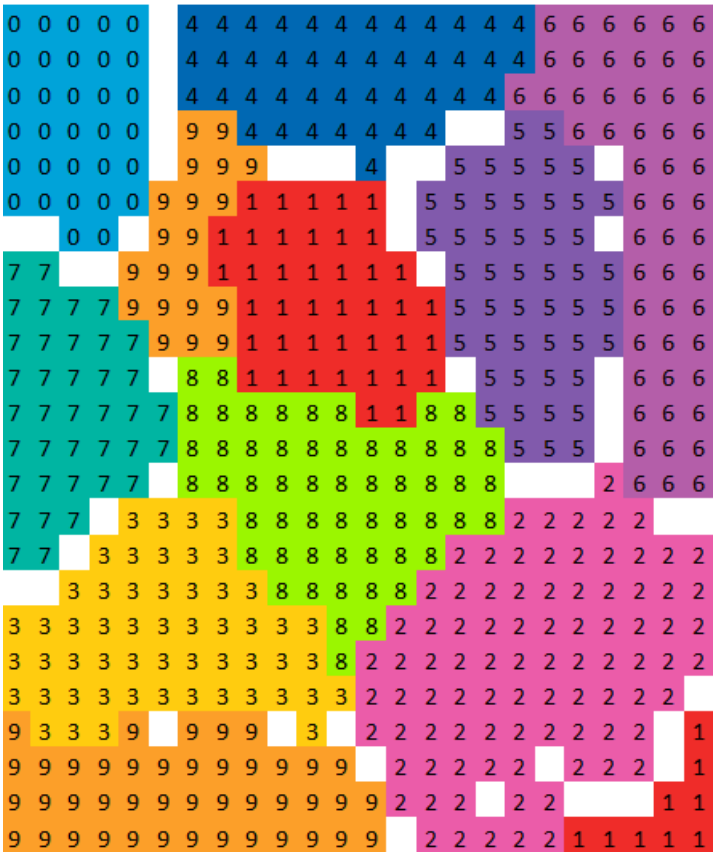
Com isso, são construídas estatísticas de precisão de reconhecimento do algoritmo implementado, que serão apresentadas a seguir.

Tabela 1. Estatísticas de precisão de reconhecimento

Dígito	Quantidade de ocorrências	% de acerto
0	87	100.00
1	97	96.91
2	92	95.65
3	85	91.76
4	114	92.11
5	108	90.74
6	87	100.00
7	96	97.92
8	91	87.91
9	89	91.01

Na Tabela 1 são apresentadas as estatísticas de precisão de reconhecimento obtidas a partir da implementação realizada e tomando como arquivo de testes *optdigits-orig.tes*. A partir da análise de seus dados percebe-se que as taxas de acerto ficaram relativamente altas e que o dígito mais difícil para a rede aprender foi o 8. Ademais, cabe destacar que a porcentagem geral de acertos nesse teste foi de 94.29%.

Figura 1. Mapa de neurônios gerado a partir da implementação realizada



Na Figura 1 é apresentado o mapa de neurônios gerado a partir da implementação realizada. Através de sua análise é perceptível que, apesar dos dígitos 9 e 1 apresentarem 2 *clusters*, em geral os dígitos ficaram agrupados em *clusters* bem definidos. Ainda, é importante destacar que após a geração dos *clusters* alguns neurônios ficaram responsáveis pelo reconhecimento de um número que não correspondia ao *cluster* em que se situavam. Então, como solicitado para a resolução do trabalho, tais neurônios foram alterados para reconhecer o número de seu *cluster* ou para não reconhecerem nenhum número (nos casos em que o neurônio se encontrava no limite entre *clusters*).

Por fim, cabe observar que a implementação realizada atendeu seu objetivo, o qual consistia na construção de uma Rede Neural de Kohonen para reconhecimento de caracteres, assim como apresentou bons resultados.

Referências

[Filho 2017] Filho, J. C. B. (2017). Notas de aula: Inteligência Artificial.

[Wikipedia 2017] Wikipedia (2017). Self-organizing map. Disponível em: https://en.wikipedia.org/wiki/Self-organizing_map. Acesso em 14/07/2017.