Trabalho 2: Implementação de Rede RBF

João Victor Garcia Coelho (Nº 10349540)
Matheus Carvalho Raimundo (Nº 10369014)
Matheus Sanchez (Nº 9081453)

Orientador: Prof. João Luís G. Rosa

USP - São Carlos

Junho/2019

Índice Analítico

1. Considerações iniciais

A sociedade está entrando em uma nova era da informação: a era da grande quantidade de dados. Com isso, começa-se a notar que precisa-se desenvolver novas formas de programação. Precisa-se de sistemas inteligentes, que possam oferecer ao usuário final uma resposta mais humana possível.

Com isso, percebe-se a grande relevância que as Redes Neurais Artificiais (RNA) têm sobre esses novos sistemas. Para fins deste projeto, foi determinado que uma RNA Radial Basis Function fosse implementada. Após, análises e estatísticas foram determinadas sobre esta rede implementada e seu treinamento através de um conjunto de dados fornecido.

2. Introdução teórica

Uma rede neural RBF é bem parecida com a MultiLayer Perceptron implementada no Trabalho 1, no que tange ao número de camadas na rede. No entanto o seu funcionamento é bem diferente.

As redes RBF são baseadas em funções de base radial (Radial Basis Function, RBF), isto é, funções que se baseiam na distância entre um ponto e um outro, denominado centro, e em um certo raio. Normalmente se usa a distância euclidiana para essas funções.

Uma rede RBF é uma rede neural artificial com uma camada de entrada, uma camada escondida e uma camada de saída. O número de neurônios em cada uma das camadas varia de problema para problema.

Para começar, as entradas são totalmente conectadas a todos os neurônios da camada escondidas. Na camada escondida, utiliza-se funções de base radial (daí o nome da rede) como funções de ativação. Como dito, essas funções se baseiam na distância de um ponto a um centro. Assim sendo, inicialmente, se utiliza algum método para escolha de centros para cada um dos neurônios da camada escondida. Pode-se usar uma escolha aleatória baseada no conjunto de treinamento, porém o método mais comum é o de clusterização conhecido como K-means. Através desse método de aprendizado não supervisionado, os centros são escolhidos por meio de clusters feitos pelo algoritmo em si.

Outro ponto fundamental para o funcionamento das RBF é a escolha do raio das funções. Esse pode ser escolhido através de várias heurísticas como: utilizar para todos os neurônios raio igual a média sobre todas as distâncias euclidianas entre o centro de cada unidade e a unidade mais próxima desta; atribuir a cada um dos raios um valor constante, geralmente, 1; cada raio ter a distância média entre o seu centro e todas as unidades. Normalmente essa escolha é feita depois de vários testes na rede.

Depois de escolhidos os parâmetros para as RBF, é necessário escolher qual função de base radial escolher. A escolha mais comum é a função gaussiana:

$$f(x) = e^{\frac{-||X-\mu||^2}{\sigma^2}}$$

onde X é o vetor ou ponto de entrada da função, μ é o centro da função (*centroid*) e σ é o raio da função. Ainda há outras opções de funções como a Multiquadrática e a Thin-Plate-Spline.

Após todas essas escolhas iniciais serem feitas, a rede pode começar a funcionar e aprender. Primeiramente os dados de entrada são, como falado anteriormente, conectados totalmente a cada um dos neurônios da camada de escondida, passam pela função de base radial, que gera uma saída. Essa saída das funções de ativação são novamente totalmente conectadas a todos os neurônios na camada de saída. Nesta última, cada uma de suas conexões possui um determinado peso w_i. Esse peso é multiplicado pela saída da função de ativação da camada escondida entrando num somatório.

Em redes RBF, as funções de ativação da camada de saída são lineares. Assim, logo após passar pela função de ativação, temos o resultado da entrada após passada pela rede.

O aprendizado da rede se dá pelo constante ajuste dos pesos das conexões entre a camada escondida e a camada de saída, geralmente feito através da regra delta. Além disso, os centros das RBF também variam e são ajustados segundo as entradas. Esse último ajuste no entanto é dado pelo algoritmos de clusterização que é não-supervisionado, diferente da Regra delta que indica um aprendizado supervisionado.

3. Conjunto de dados

O conjunto de dados utilizado para treinamento e testes desta RNA utilizado se trata de imagens/fotos de dígitos (0-9). Há também a resposta desejada para tal conjunto de dígitos.

O arquivo está em formato de texto e possui extensão **.data*. Cada linha do arquivo representa um exemplo. Em cada linha deve haver 256 números do tipo inteiro ou float, mais 10 números do tipo inteiro ou float.

Os 256 primeiros números variam entre os valores 0 e 1 (imagem monocromática) e representam todos os pixels da imagem. Os 10 últimos números variam entre os valores 0 e 1 e dizem se a imagem representa o dígito 0, 1, …, 8 ou 9 (1) ou não (0).

4. Implementação

O projeto foi implementado na linguagem Python 3, utilizando o ambiente Linux Mint. Utiliza as bibliotecas *numpy*, *sklearn* e *re*. Ao fim da implementação, tal se apresentou nos testes totalmente funcional e sem erros.

Implementadas como variáveis globais, é possível modificar o número de neurônios na camada escondida (*cluster centers*), e o limite de erro máximo permitido (que determina se um teste falhou ou sucedeu).

4.1. Entrada e manipulação

A entrada do programa se dá por meio do terminal ou linha de comandos. Ao iniciar, um menu no terminal é apresentado com as funcionalidades disponíveis. É necessário acessar a funcionalidade de treinamento antes da testagem.

4.2. Estrutura da Rede

A RNA possui fixamente 2 camadas, sendo uma escondida e uma de saída, e totalmente conectada. A camada escondida pode possuir vários neurônios (*clusters*), e a implementação permite a mudança nesse número de neurônios. Contudo, a camada de saída possui fixamente 10 neurônios, que representam cada dígito (0-9) possível.

Os pesos da camada escondida são irrelevantes na implementação do algoritmo, mas os pesos da camada de saída estão sendo representados como um array de array (2 dimensões). Na primeira dimensão, têm cada um dos neurônios da camada (0-9). Na segunda dimensão estão os pesos deste neurônio. Essa abordagem foi escolhida porque através da biblioteca *numpy* é possível fazer processamento mais rápido e fácil desses arrays/vetores. Ao início do programa, tais pesos são valores nulos (não armazenam valor efetivo), mas ao longo do treinamento eles se alteram e são armazenados.

A função de ativação fixamente escolhida foi a gaussiana:

$$f(x) = e^{\frac{-\|X-\mu\|^2}{\sigma^2}}$$

```
def gauss(value, center):
    #global sigma
    return np.exp(np.power(np.linalg.norm(value - center), 2)/np.power(5 *
sigma, 2) * -1)
```

Esta foi escolhida pois é possível aproximar qualquer tipo de função através de uma combinação de funções gaussianas, o que a torna muito bom para redes neurais que são utilizadas para classificação como a rede RBF, e mais especificamente esta utilizada no projeto.

4.3. Treinamento e aprendizado

O treinamento é realizado e os parâmetros da rede determinados utilizando o algoritmo *K-means Clustering*. Esse algoritmo funciona da seguinte maneira:

1) Determinar o número de *cluster centers* (neurônios na camada escondida) *k*.
2) Escolher aleatoriamente *k centroids* a partir do conjunto de dados de treinamento;
3) Para todos os exemplos do conjunto de treinamento, determinar o *centroid* mais próximo;
4) Para todos os *centroids*, calcular a média de todos os exemplos dentro de seu espéctro.
5) Alterar os *centroids* para os valores calculados no passo anterior (4).
6) Repetir a partir do terceiro passo (3).

Para este projeto, foi determinado por padrão um $k = 50$, após observar que tal valor é usado em algoritmos-exemplo. Além disso, é possível atualizar o *sigma* dinamicamente através da seguinte equação:

$$\sigma = \frac{d}{\sqrt{2k}}$$

onde d é a maior distância entre os *k centroids*.

Já os pesos da camada de saída, são determinados pela seguinte equação:

$$W = (G^T G)^{-1} G^T T$$

onde *G* é a saída da camada escondida (*array*), e *T* a saída final esperada (*array*).

A biblioteca *sklearn (KMeans)* foi usada para este propósito. A biblioteca *numpy* é amplamente utilizada para manipular os arrays/vetores com maior desempenho durante a implementação.

4.4. Código relevante

As linhas de código responsáveis por executar o *K-mean* são essas:

```
# K-mean
KMi = KMeans(n_clusters=N_CLUSTER_CENTERS, max_iter=100)
KMi.fit(inputs)
centroids = KMi.cluster_centers_
```

A atualização do *sigma* se dá através das linhas:

```
# Calcular sigma
distances = np.empty([N_CLUSTER_CENTERS, N_CLUSTER_CENTERS],
dtype=np.float)
for i in range(N_CLUSTER_CENTERS):
        for j in range(N_CLUSTER_CENTERS):
                distances[i][j] = np.linalg.norm(centroids[i] - centroids[j])
sigma = np.amax(distances) / np.sqrt(N_CLUSTER_CENTERS)
```

Por fim, a mudança nos pesos é feita nas linhas:

```
# Ajustar pesos
[...]
wFac = np.dot(np.linalg.inv(np.dot(distances.T, distances)), distances.T)
weights = np.dot(wFac, classes).T
```

5. Execução

Para executar o programa, é necessário instalar Python 3 e ter uma linha de comandos (terminal) minimamente regular. Para execução, use *python3 T2.py* no diretório do código *T2.py* do programa. O programa apresenta porcentagem de processamento concluído para algumas funcionalidades (‘\r’ é inserido no terminal para atualizar essa porcentagem).

5.1. Menu principal

Após iniciar o programa, o menu se apresenta ao usuário para utilização:

```
mcarvalhor@Matheus-LM-Dell:~/git/TrabalhoRedesNeurais$ python3 T2.py
Bem-vinde à nossa rede neural artificial, implementada através de uma rede RBF.

== Opções ==
1. Treinar com conjunto de dados
3. Treinar e testar com conjunto de dados (k-fold)
5. Listar pesos da camada de saída
6. Informações da Rede
0. Sair

Escolha uma opção (0-6) > 3
Entre com um arquivo com o conjunto de dados (imagens) > semeion.data
Entre com o número de blocos usados k (k) > 5

# A execução para a iteração k = 1 do 5-fold foi completa.
Acertos: 289 de 318 (90.88%).
Erro: total de 334.91 e médio de 1.05.
# A execução para a iteração k = 2 do 5-fold foi completa.
Acertos: 292 de 318 (91.82%).
Erro: total de 338.24 e médio de 1.06.
# A execução para a iteração k = 3 do 5-fold foi completa.
Acertos: 285 de 318 (89.62%).
Erro: total de 340.30 e médio de 1.07.
# A execução para a iteração k = 4 do 5-fold foi completa.
Acertos: 300 de 318 (94.34%).
Erro: total de 331.30 e médio de 1.04.
# A execução para a iteração k = 5 do 5-fold foi completa.
Acertos: 283 de 318 (88.99%).
Erro: total de 349.21 e médio de 1.10.
100.00% processado... (não pressione nenhuma tecla ainda)
# A rede aprendeu com o conjunto de dados submetido! Pesos atualizados. Além disso também foi testada.
Acertos: 1449 de 1590 (91.13%).
Erro: total de 1693.96 e médio de 1.07.

== Opções ==
1. Treinar com conjunto de dados
2. Testar com conjunto de dados
3. Treinar e testar com conjunto de dados (k-fold)
4. Testar com conjunto de dados (com detalhes)
5. Listar pesos da camada de saída
6. Informações da Rede
0. Sair

Escolha uma opção (0-6) >
```

Ao fim da execução de cada funcionalidade, este menu é novamente apresentado. Forneça entrada ao programa (aperte alguma tecla) apenas quando solicitado. Caso contrário, o programa pode ser encerrado devido à entrada mal formatada.

5.2. Funcionalidade 1

Para a funcionalidade 1 (treinamento com um conjunto de dados), é passado para o programa um arquivo no formato adequado com um conjunto de dados que serão usados apenas para treinar a rede (os pesos são alterados). O programa mistura (*shuffle*) os elementos do conjunto de dados antes de cada iteração.

5.3. Funcionalidade 2

Para a funcionalidade 2 (testagem com um conjunto de dados), é passado para o programa um arquivo no formato adequado com um conjunto de dados que serão usados apenas para testar a rede (os pesos não são alterados). Ao fim, é mostrado a estatística da execução. É necessário ter treinado a rede anteriormente para que a funcionalidade seja executada.

5.4. Funcionalidade 3

Para a funcionalidade 3 (treinamento e testagem *k-fold* com um conjunto de dados), é passado para o programa um arquivo no formato adequado com um conjunto de dados que serão usados tanto para treinar a rede (os pesos são alterados), quanto para testar a rede (os pesos não são alterados). Trata-se do algoritmo *k-fold*. São realizadas *k* iterações. O conjunto de dados fornecido é dividido em *k* partes diferentes de tamanhos iguais (exceção para casos de divisão por *k* não-exata dos dados de entrada). Dessas *k* partes do conjunto de dados, *k - 1* é utilizado para treinar a rede e 1 é utilizado para testar sobre os novos pesos em cada iteração. É possível escolher o valor *k*. O programa mistura (*shuffle*) os elementos do conjunto de dados antes de iniciar o treinamento. Ao fim, é mostrado a estatística da execução.

5.5. Funcionalidade 4

A funcionalidade 4 (testagem detalhada com um conjunto de dados), é idêntica à funcionalidade 2, com a única exceção que ela mostra também os valores esperados e produzidos para cada exemplo (qual dígito foi apropriadamente determinado). É necessário ter treinado a rede anteriormente para que a funcionalidade seja executada.

5.6. Funcionalidade 5

Para a funcionalidade 5 (listagem dos pesos da rede), é passado para o programa um neurônio dentro da camada de saída (número inteiro). O programa vai listar todos os pesos deste neurônio.

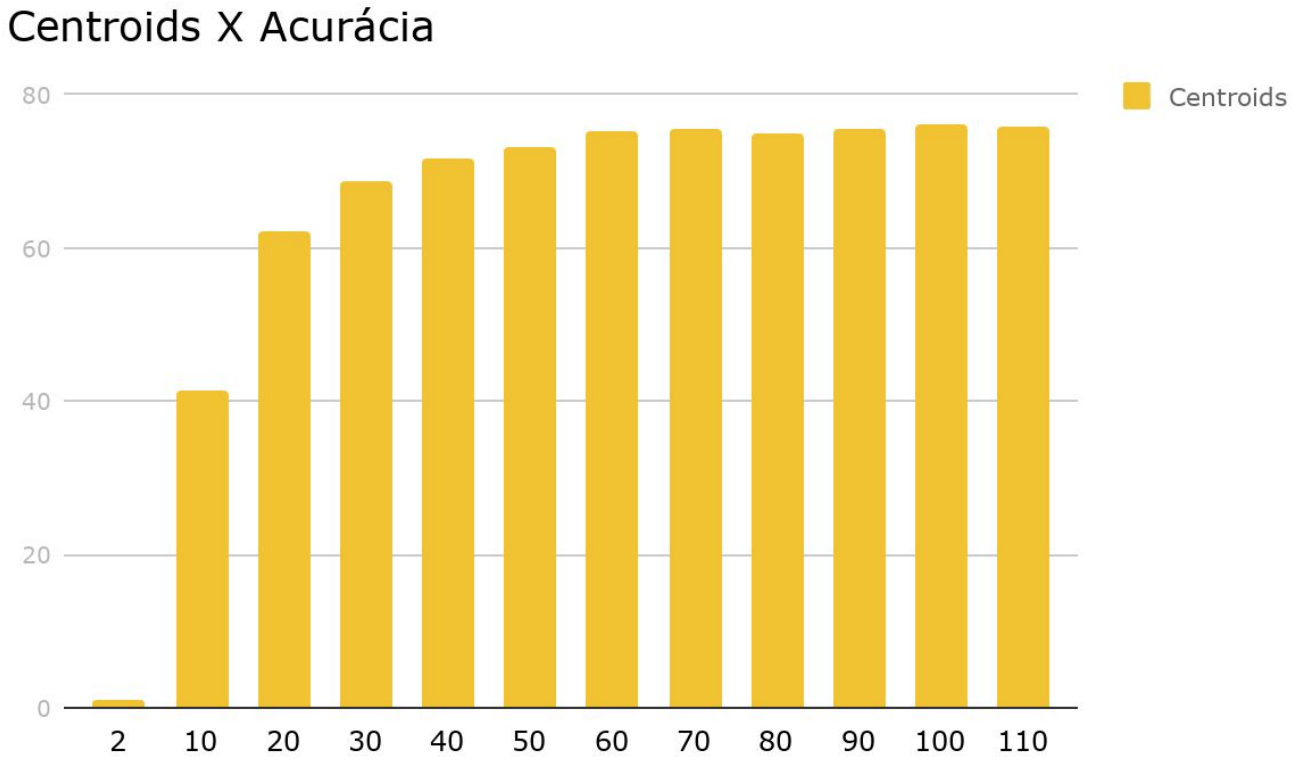
5.7. Funcionalidade 6

Para a funcionalidade 6 (informações da rede), é listado algumas informações da rede como estrutura, se já foi treinada, e caso sim, o número de exemplos que já foram usados para treinamento.

6. Estatística e discussão

Para a avaliação do modelo projetado, foi o utilizado a funcionalidade 3, validação cruzada junto com 5 folds, gerando diversas estatísticas.

Variando o número de neurônios na camada escondida (*centroids*), obtemos o seguinte gráfico:



Pode ser observado que com um número de neurônios entre $50 \leq k \leq 110$, a acurácia é maior.

7. Conclusão

Neste projeto, foi possível chegar à algumas conclusões. A primeira e a mais intuitiva delas é que as redes neurais RBF apresentam desempenho no tempo de treinamento superior à redes MLP. O principal motivo pelo qual isso ocorre se dá por conta do algoritmo *K-mean Clustering*, que pode ser executado em poucas interações, quando comparado com o *Error Back-propagation* sozinho.

Também pode-se concluir que, apesar do maior desempenho, esta implementação da Rede RBF apresentou menor acurácia quando comparada com a implementação da Rede MLP para o mesmo projeto (Trabalho 1). É possível que isso tenha ocorrido por conta da escolha dos parâmetros da rede, como o *k*, e o algoritmo *K-mean*.

Uma outra conclusão à qual chegou-se também, é que alterar o número de neurônios na camada escondida (*cluster centers*), na forma como implementou-se o projeto, é possível e fácil: apenas mudando o valor de uma variável no código.

Por fim, pode-se dizer que os conceitos vistos em sala de aula na teoria foram colocados na prática com esse projeto. E com a prática, não apenas algumas questões/dúvidas foram solucionadas, como também enxergou-se melhor a diferença de implementação e funcionamento da Rede RBF para a Rede MLP.

8. Bibliografia

Material de aula do professor disponibilizado.

Radial Basis Function Network. Disponível em <https://www.hackerearth.com/blog/developers/radial-basis-function-network/>. Acessado em 12 de junho de 2019.