Implementação de uma Rede Neural para Reconhecimento de Dígitos Manuscritos em Imagens Utilizando Octave

Alex Seródio Gonçalves

Objetivos

- Desenvolver uma rede neural artificial que reconheça dígitos manuscritos em imagens de 28x28 pixels (784 pixels);
- Treinar a rede de forma a reconhecer os dígitos manuscritos;
- Utilizar a linguagem de programação Octave na implementação;
- Utilizar a base de dados MNIST no treinamento da rede.

Octave (1993)

- Octave é uma linguagem de programação interpretada de domínio científico e de licença livre;
- Desenvolvida principalmente por John W. Eaton;
- Possui ferramentas extensivas para resolver problemas comuns de álgebra linear, com foco principal na computação de problemas lineares e não lineares;
- Foi originalmente concebida para substituir o uso de linguagens como Fortran na resolução de problemas de engenharia química na universidade de Wisconsin-Madison.

Redes Neurais Artificiais

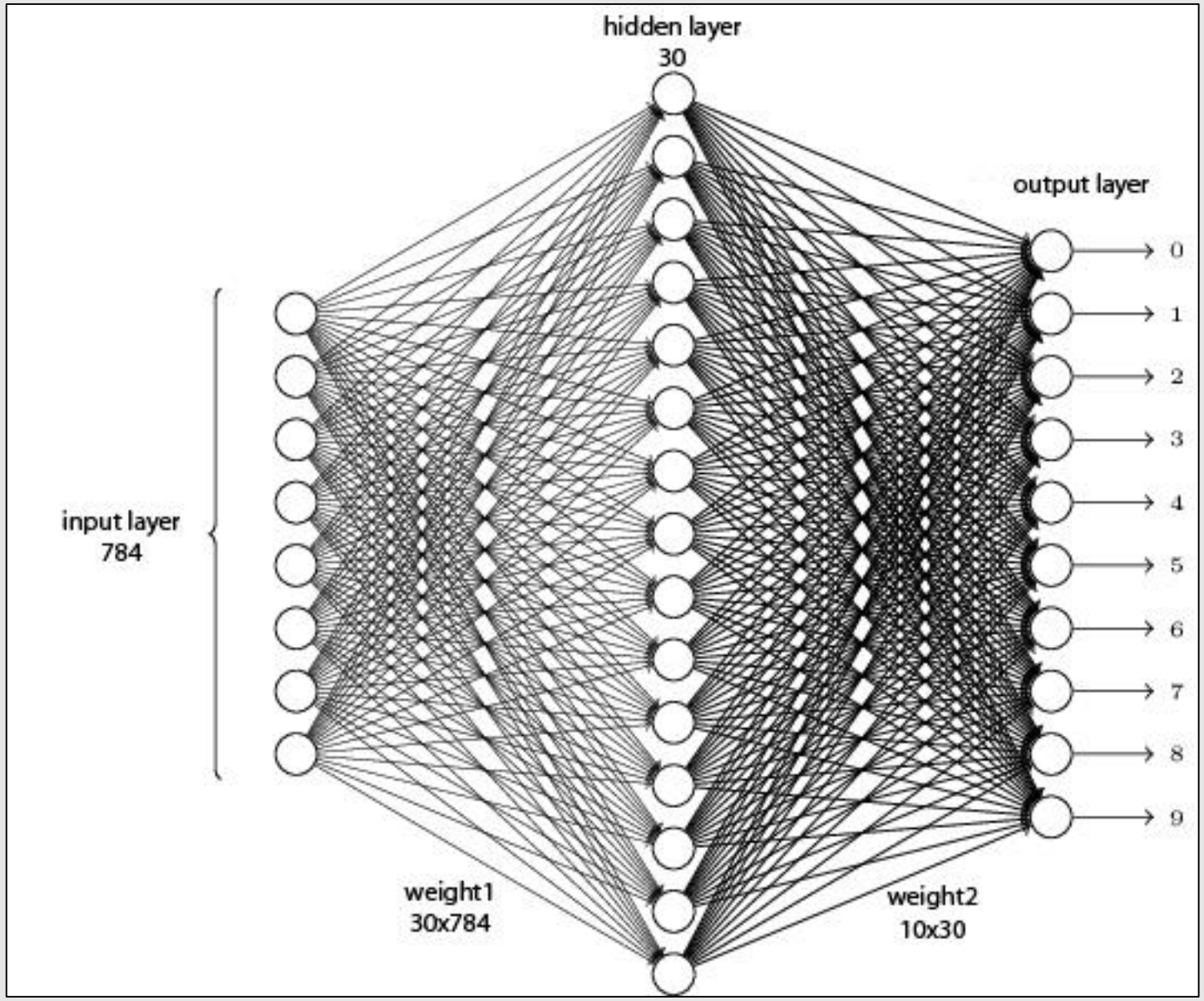
Computacionalmente, cada camada é representada por um vetor e cada conjunto de pesos é representado por uma matriz.

Função *feedforward:*

$$a^2 = W a^1 + b$$

Função sigmoid:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Implementação: *feedforward* e *sigmoid*

```
1 function result = sigmoid (x)
2   result = 1.0 ./ (1.0 + exp(-x));
3 endfunction
```

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

```
1 function output = feed_forward (input, bias2, bias3, weight1, weight2)
2   z = (weight1 * input) + bias2;
3   output = sigmoid(z);
4   z = (weight2 * output) + bias3;
5   output = sigmoid(z);
6 endfunction
```

$$a^2 = Wa^1 + b$$

Observações:

- Variáveis com tipagem dinâmica;
- Retorno especificado na assinatura da função;
- Multiplicação matricial por padrão.

Implementação: inicialização da rede

```
layers_size = [784, 30, 10];
epochs = 30;
batch_size = 10;
learning_rate = 3.0;
[bias2, bias3, weight1, weight2] = stochastic_gradient_descent(training_images,
          training_values, layers_size, epochs, batch_size, learning_rate);
```

Observações:

- *layers_size*: lista com os tamanhos de cada uma das três camadas;
- *epochs*: quantidade de seções de treino;
- *batch_size*: intervalo de atualização da rede;
- *learning_rate*: a importância das alterações realizadas;

Implementação: *stochastic_gradient_descent*

```
1  function [b2, b3, w1, w2] = stochastic_gradient_descent (trainX, trainY,
2           layers_size, epochs, batch_size, lea)
3
4    for j = 1:epochs
5      for i = 1:batch_size+1:training_size-batch_size
6        batchX = training_dataX(:, [i:batch_size+i]);
7        batchY = training_dataY(:, [i:batch_size+i]);
8        [b2, b3, w1, w2] = update_network(batchX, batchY, lea, b2, b3, w1, w2);
9      endfor
10   endfor
11 endfunction
```

Observações:

- Índices começam em 1 e não em 0;
- Estrutura do for (início : passo : final);
- Acesso a vários elementos de uma lista;
- Funções também podem retornar uma lista de valores;

Implementação: *update_network*

```
function [bias2, bias3, weight1, weight2] = update_network (batchX, batchY, learning_rate,
      bias2, bias3, weight1, weight2)

  for x = 1:batches
    [d_b2, d_b3, d_w1, d_w2] = backpropagation(mini_batchX(:,x), mini_batchY(:,x),
          bias2, bias3, weight1, weight2);

    w1 += d_w1;
    w2 += d_w2;
    b2 += d_b2;
    b3 += d_b3;
  endfor

   weight1 = weight1 - ((learning_rate / batches) * w1);
   weight2 = weight2 - ((learning_rate / batches) * w2);
   bias2 = bias2 - ((learning_rate / batches) * b2);
   bias3 = bias3 - ((learning_rate / batches) * b3);
endfunction
```

Implementação: *backpropagation*

```
function [b2, b3, w1, w2] = backpropagation(x, y, bias2, bias3, weight1, weight2)

  activation = x;
  activations{1} = x;

  z = (weight1 * activation) + bias2;
  zs{1} = z;
  activation = sigmoid(z);
  activations{2} = activation;

  z = (weight2 * activation) + bias3;
  zs{2} = z;
  activation = sigmoid(z);
  activations{3} = activation;

  delta = (activations{3} - y) .* sigmoid_derivative(zs{2});
  b3 = delta;
  w2 = delta * activations{2}';

  z = zs{1};
  sp = sigmoid_derivative(z);
  delta = (weight2' * delta) .* sp;
  b2 = delta;
  w1 = delta * activations{1}';
endfunction
```

Observações:

- Cell Arrays;
- Matriz transposta;

Implementação: *check_input*

```
1  function answer = check_input (weight1, weight2, bias2, bias3, image_path)
2
3    image = double(imread(image_path));
4    image = sum(255 - image, 3);
5    image = image / 255;
6    input = reshape(image,1,[])';
7
8    z = weight1 * input + bias2;
9    output = sigmoid(z);
10   z = weight2 * output + bias3;
11   output = sigmoid(z);
12
13   [max, answer] = max(output);
14   answer -= 1;
15   imagesc(reshape(input, 28, 28));
16   printf("Hipotese: %d\n", answer);
17 endfunction
```

Conclusões e Discussões

- A rede atingiu uma taxa de em média 94% de precisão;
- Boa documentação;
- Boa legibilidade e facilidade de escrita;
- Possibilitou o estudo de uma linguagem e de um paradigma de programação ainda não conhecidos pelo autor.

Referências

- Nielsen, M. (2017) “Neural Networks and Deep Learning”, http://neuralnetworksanddeeplearning.com/index.html, Junho.
- GOLDSCHMIDT, R. R. Uma introdução à Inteligência Computacional: Fundamentos, Ferramentas e Aplicações. 1ª edição.
- Octave (2018) “GNU Octave”, https://www.gnu.org/software/octave/, Junho.
- LeCun Y., Cortes C., Burges C. J. C. (XXXX) THE MNIST DATABASE of handwritten digits, http://yann.lecun.com/exdb/mnist/, Junho.
- John W. Eaton (2018) “GNU Octave”, https://octave.org/doc/v4.2.2/, Junho.