

# Exercício programa - Reconhecimento de dígitos

## Computação III - CCM0218

Carlo Bellinati NUSP: 11258360

Rafael Badain NUSP: 10277102

28/11/2020

### 1 Primeira tarefa

Na primeira tarefa, o objetivo era implementar um programa que resolvesse múltiplos sistemas lineares através de fatoração QR, realizada por sucessivas rotações de Givens.

A rotação de Givens está implementada em **rot\_givens.py**, e a resolução de sistemas lineares com fatoração QR está implementada em **systems\_qr.py**. Fizemos os testes pedidos para a resolução dos sistemas únicos  $Wx = b$  (testes A e B) e os sistemas múltiplos  $WH = A$  (testes C e D):

- Teste A (Sistema único e determinado):  $n = m = 64, W_{i,i} = 2, i = 1, \dots, n, W_{ij} = 1$  se  $|i - j| = 1$  e  $W_{ij} = 0$  se  $|i - j| > 1$  e  $b(i) = 1, i = 1, \dots, n$
- Teste B (Sistema único e sobredeterminado):  $n = 20, m = 17, W_{ij} = 1/(i + j - 1)$  se  $|i - j| < 4$  e  $W_{ij} = 0$  se  $|i - j| > 4$  e  $b(i) = i, i = 1, \dots, n$
- Teste C (Sistema múltiplo e determinado):  $n = m = 64, W_{i,i} = 2, i = 1, \dots, n, W_{ij} = 1$  se  $|i - j| = 1$  e  $W_{ij} = 0$  se  $|i - j| > 1$  e  $A(i, 1) = 1, A(i, 2) = i, A(i, 3) = 2i - 1, i = 1, \dots, n$
- Teste D (Sistema múltiplo e sobredeterminado):  $n = 20, m = 17, W_{ij} = 1/(i + j - 1)$  se  $|i - j| < 4$  e  $W_{ij} = 0$  se  $|i - j| > 4$  e  $A(i, 1) = 1, A(i, 2) = i, A(i, 3) = 2i - 1, i = 1, \dots, n$

As matrizes com as soluções dos sistemas dos testes A a D podem ser encontradas em **testes\_primeira\_tarefa.txt**. Foi calculado em cada caso o erro quadrático da resolução desses sistemas, isto é, a norma euclidiana da matriz  $B = A - WH$ , dada por

$$E = \|B\| = \sqrt{\sum_{i=0}^n \sum_{j=0}^m B(i, j)^2}$$

Nos testes A e C, que resolviam sistemas determinados ( $n = m$ ), temos um erro igual a zero, como era esperado. Para os testes B e D, que resolvem sistemas sobredeterminados ( $n > m$ ), o resultado já mostra um erro significativo de, respectivamente, 11.153786357262028 e 24.466302973737676.

## 2 Segunda tarefa

Na segunda tarefa tínhamos que implementar um código que realizasse uma fatoração não negativa de uma matriz  $A$   $n \times m$  em matrizes  $W$   $n \times p$  e  $H$   $p \times m$ . Essa fatoração não é necessariamente exata e apresenta um erro quadrático que o algoritmo busca minimizar. O resultado também pode não convergir, por isso estabelecemos um número máximo de iterações e um limite de erro para declarar convergência do resultado da fatoração. O código com essa implementação está em **nnmf.py**. Usamos o exemplo do roteiro para decompor a seguinte matriz:

$$A = \begin{pmatrix} 3/10 & 3/5 & 0 \\ 1/2 & 0 & 1 \\ 4/10 & 4/5 & 0 \end{pmatrix}$$

Os resultados obtidos estão em **testes\_segunda\_tarefa.txt**. Para cada vez que rodamos o programa, duas diferentes decomposições,  $WH$  e  $W'H'$ , eram possíveis de serem obtidas:

$$W = \begin{pmatrix} 3/5 & 0 \\ 0 & 1 \\ 4/5 & 0 \end{pmatrix} \quad H = \begin{pmatrix} 1/2 & 1 & 0 \\ 1/2 & 0 & 1 \end{pmatrix}$$

$$W' = \begin{pmatrix} 0 & 3/5 \\ 1 & 0 \\ 0 & 4/5 \end{pmatrix} \quad H' = \begin{pmatrix} 1/2 & 0 & 1 \\ 1/2 & 1 & 0 \end{pmatrix}$$

Podemos observar que  $WH = W'H' = A$ , ou seja, ambas as decomposições são exatas e portanto tem erro zero. Esse problema não tem solução única, pois ao resolver o problema de mínimos quadrados, que minimiza o erro quadrático, se há mais de uma solução exata, há mais de uma solução possível, e dependendo dos valores iniciais de  $W$  (que são aleatórios) podemos obter resultados diferentes.

Testamos também a frequência de cada uma dessas decomposições e verificamos que ambas decomposições acontecem na mesma proporção (50% cada). Todos os testes foram realizados com os mesmos parâmetros de convergência:  $itmax = 100$  e  $\epsilon = 10^{-5}$ .

## 3 Tarefa principal

### 3.1 Fase de treinamento

Nessa fase do projeto, realizamos o aprendizado de cada dígito  $d$ , que é armazenado em uma matriz  $W_d$   $784 \times p$ .  $W_d$  é uma matriz que contém  $p$  imagens de tamanho  $28 \times 28$ . Essas imagens são como um "resumo" das  $ndig\_treino$  imagens do dígito  $d$  analisadas de um banco de dados. A estratégia para se obter esse "resumo" de muitas imagens feito de poucas é utilizar a fatoração não negativa. Seja  $A$  uma matriz  $784 \times ndig\_treino$  que contém as imagens do banco de dados usadas para o treinamento, obtemos  $W_d$  fazendo a fatoração não negativa  $W_d H = A$ . Esse código está no arquivo **training\_MNIST.py**.

Para os dígitos de 0 a 9, usamos os valores de referência  $ndig\_treino = 100, 1000, 4000$  e  $p = 5, 10, 15$ . Cada matriz classificadora está salva na pasta **output** com o nome **W\_d\_ndigtreino\_p.txt**.

Uma análise importante é a do tempo gasto para treinar cada matriz  $W_d$ . A tabela 1 mostra a média, em segundos, do tempo de execução dos dígitos em cada caso de  $ndig\_treino$  ( $ndt$ ) e  $p$ .

$ndt/p$	5	10	15
100	8,896	20,872	31,816
1000	25,761	50,892	77,337
4000	75,706	154,249	232,301

Tabela 1: Médias dos tempos de execução dos dígitos

Analisando o comportamento de  $p$ , vemos que, para todos  $ndig\_treinos$ , o tempo, aproximadamente, sempre é somado a um mesmo valor ao se adicionar 5 unidades ao valor de  $p$ . O tempo dobra de  $p = 5$  para  $p = 10$  e é multiplicado por 1.5 quando  $p$  vai de  $p = 10$  para  $p = 15$ .

Já em relação ao comportamento do tempo de execução, em função de  $ndig\_treino$ , temos que o tempo médio de execução, aproximadamente, duplica de  $ndt = 100$  para  $ndt = 1000$  e triplica de  $ndt = 1000$  para  $ndt = 4000$ .

Podemos também analisar o tempo de execução de cada dígito individualmente. A tabela 2 mostra os tempos de execução  $t(d, ndig\_treino, p)$  para treinar cada dígito  $d$  para cada par  $(ndig\_treino, p)$ . A última coluna mostra uma média normalizada do tempo de execução de todos os casos  $\bar{t}_d$ , calculada por

$$\bar{t}_d = \frac{S_d}{\sum_{D=0}^9 S_D}, \quad S_d = \sum_{(n,p)} \frac{t(d, n, p)}{\sum_{D=0}^9 t(D, n, p)}$$

O valor dessas médias também é representado pelo gráfico da figura 1. Podemos perceber que, para cada caso, os dígitos mais demorados e mais rápidos mudam, e a diferença entre eles é baixa (desvio padrão pequeno). No gráfico vemos que, em uma média geral, o dígito mais rápido é o 5 enquanto os mais demorados são o 0, 3 e 9.

## 3.2 Classificação dos dígitos

A última etapa do projeto era a de classificação dos dígitos. Temos um banco de imagens que devem ser testadas e classificadas entre os dígitos de 0 a 9, usando as matrizes que representam o aprendizado de máquina,  $W_d$ . Existe um gabarito para esse teste, e com ele calculamos a porcentagem de acerto geral ( $P = n\_acertos/n\_teste$ ) e a porcentagem de acerto para cada dígito ( $P_d = n\_acertos_d/n\_teste_d$ ).

O programa faz isso resolvendo um problema de mínimos quadrados, que são sistema simultânea sobreterminadoa  $W_d H = A$ , onde  $A$  é uma matriz  $784 \times n\_teste$  que contém as imagens a serem testadas. Verificamos para cada um desses sistemas, com qual matriz  $W_d$  ocorreu o menor erro, e então classificamos esse dígito como  $d$ . O código está presente no arquivo **classify\_MNIST.py** e todas as classificações foram feitas usando  $n\_teste = 10000$ . Os resultados descritos a seguir estão na pasta outputs no arquivo **classify\_index\_10000.txt**.

Vamos analisar primeiro o valor da porcentagem de acerto em função de  $ndig\_treino$  e  $p$ . A tabela 3 mostra a porcentagem geral de acerto para cada  $ndig\_treino$  e  $p$ . Como esperado, quando aumentamos  $p$  e  $ndig\_treino$ , a precisão cresce. Os efeitos

de aumento de  $p$  e  $ndig\_treino$ , para os valores dados, parecem ter uma contribuição semelhante.

$d/(n,p)$	(100,5)	(100,10)	(100,15)	(1000,5)	(1000,10)
0	9,83	22,97	30,39	26,4	51,41
1	9,93	22,14	29,77	25,01	49,57
2	9,19	21,68	30,25	24,73	40,71
3	9,26	20,67	32,48	28,98	50,98
4	9,51	19,32	33,31	25,51	50,56
5	5,44	19,64	30,54	24,98	49,8
6	9,72	19,13	31,79	25,01	48,3
7	6,64	20,72	34,24	25,62	53,46
8	9,22	21,28	32,09	24,79	50,81
9	10,12	21,1	31,3	26,13	54,21

$d/(n,p)$	(1000,15)	(4000,5)	(4000,10)	(4000,15)	Valor médio
0	76,72	75,25	156,57	226,91	0,102242341
1	77,05	76,36	156,12	231,34	0,1010877126
2	76,17	75,26	160,82	241,14	0,09851495129
3	77,05	75,13	152,8	234,04	0,1021577418
4	76,6	75,12	158,84	229,93	0,1006228557
5	79,31	75,63	154,77	233,76	0,09468696881
6	75,82	74,71	146,98	226,81	0,09835575354
7	80,37	75,66	151,25	235,58	0,09914328646
8	75,94	75,69	154,37	234,13	0,1004874362
9	78,19	78,15	149,89	229,22	0,1027009527

Tabela 2: Tempos de execução para cada dígito

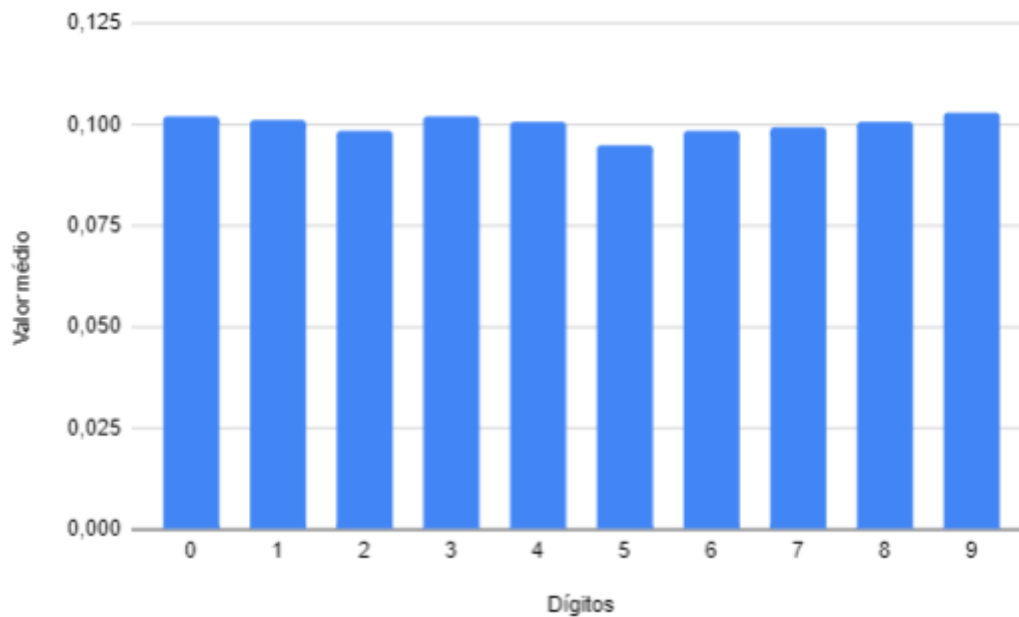


Figura 1: Valores médios normalizados do tempo de execução para cada dígito

ndt/p	5	10	15
100	88,01	90,06	90,95
1000	90,75	92,91	93,59
4000	91,69	93,49	93,92

Tabela 3: Porcentagens gerais de acerto

Podemos comparar as tabelas 1 e 3. As tabelas 4 e 5 mostram a ordem crescente dos termos das tabelas 1 e 3, respectivamente. De modo geral, o crescimento do tempo de treinamento é acompanhado de um aumento na porcentagem de acertos. Porém, foi possível notar alguns casos que a precisão era maior e o tempo de treinamento menor. São os casos das células em verde nas tabelas, que se mostram mais vantajosas que as células vermelhas. Por exemplo, é mais rápido e mais preciso usar  $(ndig\_treino, p) = (1000, 10)$  do que  $(4000, 5)$ . Da mesma forma, é mais vantajoso usar  $(1000, 15)$  do que  $(4000, 10)$ . Teríamos que realizar mais testes para conferir esse padrão, mas ele parece favorecer o aumento de  $p$  em relação ao aumento de  $ndig\_treino$ .

ndt/p	5	10	15
100	1	2	4
1000	3	5	7
4000	6	8	9

Tabela 4: Ordem crescente da tabela 1

ndt/p	5	10	15
100	1	2	4
1000	3	6	8
4000	5	7	9

Tabela 5: Ordem crescente da tabela 3

Podemos agora, analisar os dígitos separadamente, avaliando os valores de  $P_d$  para cada par  $(ndig\_treino, p)$  com  $d = 0, \dots, 9$ . A tabela 6 possui esses valores. O gráfico da figura 2 mostra valores médios normalizados para a precisão de cada dígito para cada par  $(ndig\_treino, p) = (n, p)$  e  $d = 0, \dots, 9$

$$\bar{P}_d = \sum_{(n,p)} \frac{P_d(n,p)}{\sum_{D=0}^9 P_D(n,p)}$$

A tabela 6 mostra que existe uma clara disparidade entre a precisão de cada dígito. Existem dígitos com precisão consideravelmente mais alta que outros, como por exemplo, o dígito 1 e o dígito 8. Essas diferenças se mantêm para todos  $(ndig\_treino, p)$ , isto é, a ordem dos dígitos em termos da precisão é razoavelmente constante.

Os dígitos cuja classificação é mais precisa são 0 e 1, e os dígitos das mais imprecisas são o 5 e 8. Aparentemente, não há muita semelhança com o padrão de tempo de treinamento exibido no gráfico da figura 1. Isso nos leva a supor que um dígito pode ser mais "simples" quanto a sua classificação, mas mais "complexo" quanto ao seu treinamento, e vice-versa.

d/(n,p)	(100,5)	(100,10)	(100,15)	(1000,5)	(1000,10)
0	97,55	97,65	97,85	97,55	98,67
1	99,21	99,47	99,47	99,20	99,55
2	84,78	90,11	90,01	86,91	90,79
3	85,04	86,13	87,82	90,49	91,98
4	80,24	84,41	87,16	86,55	92,97
5	83,07	86,54	84,30	86,32	89,57
6	93,00	96,34	95,51	90,03	95,92
7	88,42	91,34	94,26	89,49	90,85
8	79,87	79,26	82,23	86,96	87,78
9	86,81	87,71	88,99	87,11	89,99

d/(n,p)	(1000,15)	(4000,5)	(4000,10)	(4000,15)
0	98,77	97,65	98,87	98,77
1	99,55	99,38	99,38	99,38
2	92,34	89,92	91,66	92,63
3	92,07	92,47	93,76	93,06
4	93,27	88,48	92,56	93,48
5	90,02	88,00	90,02	90,35
6	94,45	96,76	96,13	97,18
7	93,09	88,81	91,34	90,66
8	87,37	88,80	89,52	91,06
9	91,87	85,53	90,68	91,77

Tabela 6: Porcentagens de acerto de cada dígito

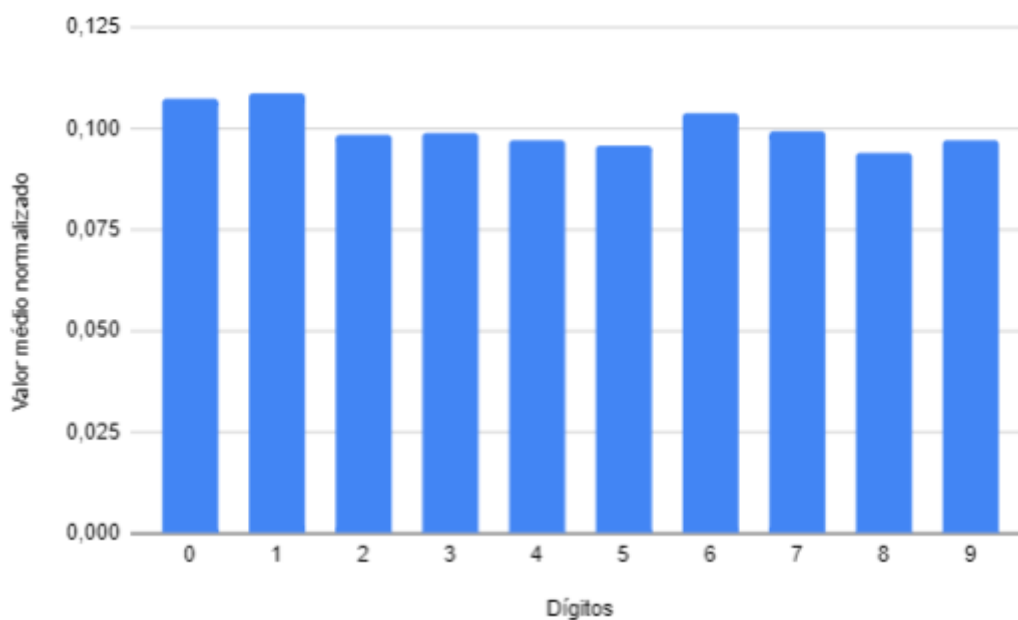


Figura 2: Valores médios normalizados da precisão para cada dígito