Resolução de Sistema de Equações Lineares de Matrizes Pentadiagonais com Redes Neurais Convolucionais*

Tiago C A Amorim (RA: 100675)[a], Taylon L C Martins (RA: 177379)[b]

[a] *Doutorando no Departamento de Engenharia de Petróleo da Faculdade de Engenharia Mecânica, UNICAMP, Campinas, SP, Brasil*
[b] *Aluno especial, UNICAMP, Campinas, SP, Brasil*

Keywords: Rede Neurais Convolucionais, Sistemas de Equações Lineares

1. Introdução

2. Motivação

O método das diferenças finitas é utilizado para resolver diferentes problemas físicos que podem ser descritos como equações diferenciais. O método se baseia na aproximação das derivadas por diferenças finitas (exemplos em 2.1). O domínio espaço-temporal é discretizado e a solução das equações diferenciais é aproximada nos nós desta malha [1]. O problema então é transformado em uma série de equações não-lineares, que usualmente são resolvidas por métodos numéricos, como o método de Newton-Raphson [2].

$$\frac{\partial u(x)}{\partial x} \approx \frac{u(x+h) - u(x-h)}{2h} \tag{2.1a}$$

$$\frac{\partial^2 u(x)}{\partial x^2} \approx \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} \tag{2.1b}$$

É usual utilizar aproximações de derivada que utilizam os valores da função do nó e seus vizinhos diretos. Em problemas bidimensionais com malhas regulares (Figura 1) esta escolha leva a sistemas de equações com matrizes pentadiagonais (exemplo para uma malha n_i,n_j em 2.2). Cada equação não-linear tem termos associados a um dos nós (i,j) e seus quatro vizinhos: (i-1,j), (i+1,j), (i,j-1) e (i,j+1). A resolução numérica deste sistema de equações não-lineares usualmente está associada a métodos iterativos, em que novas equações lineares são resolvidas. Desta forma, a resolução do problema original está associada à solução de um significativo número de sistemas de equações lineares com matriz pentadiagonal.

A proposta deste trabalho é avaliar a possibilidade de utilizar uma rede convolucional para resolver este tipo de sistema de equações lineares.

Figura 1: Esquemático de uma malha regular, com destaque para a célula (i,j) e seus vizinhos.

	(i,j-1)		
(i-1,j)	**(i,j)**	(i+1,j)	
	(i,j+1)		

3. Trabalhos Correlatos

Não foram encontrados muitos trabalhos com foco na resolução de sistemas de equações lineares com redes neurais. Duas formas distintas de resolução do problema foram propostas. A primeira vertente é resolver o sistema de equações lineares junto com o treinamento da rede ([3]). Uma aplicação interessante desta proposta é o de resolver sistema de grande dimensão, que possivelmente não cabem na memória disponível, e usar a rede para aprender um mapeamento que aproxima a resposta ([4]).

Uma segunda vertente é a de treinar uma rede neural com base em vários exemplos de sistemas de equações a resolver. A rede treinada é utilizada para resolver novos sistemas de equações. Uma proposta focou na solução de sistemas tridiagonais ([5]), utilizando uma série de camadas densas seguidas por conexões residuais. Um outro trabalho ([6]) foca na solução de problemas físicos associados a equações diferenciais. Este trabalho tenta primeiro encontrar uma representação densa dos dados de entrada por meio de uma rede *autoenconder*. Posteriormente a representação densa de cada amostra passa por uma outra rede neural que busca resolver o problema.

A proposta estudada neste projeto segue a segunda vertente. É feita a opção de utilizar camadas convolucionais para que a arquitetura da rede seja agnóstica à discretização do problema (tamanho da malha).

4. Codificação do Sistema Linear

O objetivo da rede é resolver um problema do tipo $\mathbf{Ax} = \mathbf{b}$. Cada amostra da base de dados são os valores

*Projeto final como parte dos requisitos da disciplina IA048: Aprendizado de Máquina.

$$\overbrace{\begin{bmatrix} a_1^0 & a_1^1 & 0 & \dots & 0 & a_1^{n_i} & 0 & & \dots & 0 \\ a_2^{-1} & a_2^0 & a_2^1 & 0 & \dots & 0 & a_2^{n_i} & 0 & \dots & 0 \\ 0 & a_3^{-1} & a_3^0 & a_3^1 & 0 & \dots & 0 & a_3^{n_i} & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & & & & \ddots & \vdots \\ 0 & \dots & 0 & a_{n_in_j-2}^{-n_i} & 0 & \dots & a_{n_in_j-2}^{-1} & a_{n_in_j-2}^{0} & a_{n_in_j-2}^{1} & 0 \\ 0 & \dots & & 0 & a_{n_in_j-1}^{-n_i} & 0 & \dots & a_{n_in_j-1}^{-1} & a_{n_in_j-1}^{0} & a_{n_in_j-1}^{1} \\ 0 & \dots & & & 0 & a_{n_in_j}^{-n_i} & 0 & \dots & a_{n_in_j}^{-1} & a_{n_in_j}^{0} \end{bmatrix}}^{\mathbf{A}} \overbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n_in_j-2} \\ x_{n_in_j-1} \\ x_{n_in_j} \end{bmatrix}}^{\mathbf{x}} = \overbrace{\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n_in_j-2} \\ b_{n_in_j-1} \\ b_{n_in_j} \end{bmatrix}}^{\mathbf{b}} \tag{2.2}$$

das diagonais da matriz **A** e o vetor **b**, e a saída pretendida são os valores de **x**. Para simplificar a quantidade de dados a serem repassados à rede, é possível dividir as linhas da matriz **A** e o vetor **b** pelos valores da diagonal principal. Esta opração não muda o **x** que resolve o sistema. Desta forma que na nova matriz pentadiagonal todos os valores da diagonal principal são iguais à unidade.

Os dados são organizados em forma de tensor 2D, à semelhança de uma imagem com (n_i,n_j) pixels. Seguindo a notação utilizada em 2.2, os *canais* desta imagem correspondem aos valores das diagonais $-n_i$, -1, 1 e n_i, e os valores de **b**. Cada um destes vetores é ajustado para que os termos fiquem na posição (i,j) correspondente ao nó associado ao valor.

A saída pretendida, **x**, também é formatada como um tensor 2D (com apenas um canal). Desta forma, a rede neural recebe uma *imagem* (n_i,n_j) com 5 canais e deve gerar uma imagem de mesmo tamanho com 1 canal.

5. Arquitetura da Rede Convolucional

A arquitetura da rede foi construída de forma a utilizar apenas camadas que

A rede neural é treinada com amostras geradas aleatoriamente. A cada solicitação de uma nova amostra são gerados a matrix **A** e o vetor **x**. O vetor **b** é a multiplicação matricial dos dois primeiros termos. Em seguida os dados são reoganizados em forma de tensores.

6. Resultados

Todo o código foi desenvolvido em Pytorch, e está disponível em https://github.com/TiagoCAAmorim/machine_learning/blob/main/Projeto/solve_lin_eq.ipynb.

7. Conclusão

Referências

[1] D. Causon, C. Mingham, Introductory finite difference methods for PDEs, Bookboon, 2010.

[2] R. L. Burden, J. D. Faires, A. M. Burden, Análise numérica, Cengage Learning, 2016.

[3] A. Cichocki, R. Unbehauen, Neural networks for solving systems of linear equations and related problems, IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications 39 (2) (1992) 124–138.

[4] Y. Gu, M. K. Ng, Deep neural networks for solving large linear systems arising from high-dimensional problems, SIAM Journal on Scientific Computing 45 (5) (2023) A2356–A2381.

[5] Z. Jiang, J. Jiang, Q. Yao, G. Yang, A neural network-based pde solving algorithm with high precision, Scientific Reports 13 (1) (2023) 4479.

[6] K. Kontolati, S. Goswami, G. Em Karniadakis, M. D. Shields, Learning nonlinear operators in latent spaces for real-time predictions of complex dynamics in physical systems, Nature Communications 15 (1) (2024) 5101.