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.

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

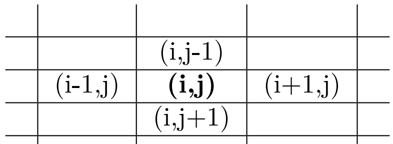

Figura 1: Esquemático de uma malha regular, com destaque para a célula (i,j) e seus vizinhos.

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

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

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 operação não muda o **x** que resolve o sistema. Desta forma 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 e Treinamento

Como as *resolução* dos dados de entrada e saída de rede são as mesmas, optou-se por utilizar apenas camadas que mantém este tamanho. A rede é composta por (Figura 2):

1. Camada convolucional com *kernel*=1x1: passa de 5 para n_{lat} o número de canais (ativação: ReLu).
2. N camadas convolucionais com *kernel*=3x3 e *padding*=1 (ativação: ReLu).
3. Camada convolucional com *kernel*=1x1: passa de n_{lat} para um canal.

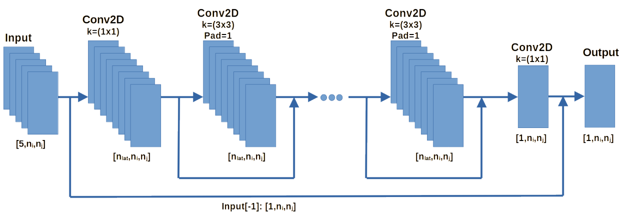

Figura 2: Arquitetura geral da rede neural proposta.

A ligação direta entre os dados de entrada e a saída é dos valores do vetor **b** (último *canal* de cada amostra).

A classe que constrói a rede neural tem diferentes opções de configuração, de forma a poder ser feita uma otimização destes hiperparâmetros. Entre as opções de arquitetura, existe a possibilidade de trocar as camadas convolucionais com *kernel*=3x3 por blocos **Inception** (Figura 3).

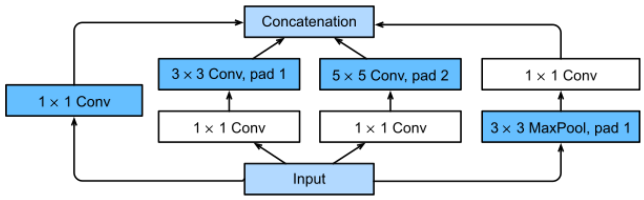

Figura 3: Bloco Inception. (Fonte: [7])

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.

Para tornar a rede mais generalizável, diferentes distribuições probabilísticas são empregadas na geração de **A** e **x**, e com todos os valores *aproximadamente* no intervalo $[-2, 2]$. A cada época de treinamento são apresentados 10 000 amostras, em *batches* de 64. Como a geração das amostras é aleatória, não foi necessário dividir a base de dados em treinamento, validação e teste, pois toda amostra é *inédita* para a rede.

A função de perda é o RMSE (Equação 5.1). O treinamento da rede é feito com o algoritmo Adam, com taxa de aprendizado inicial de 0.01. O passo de treinamento é reduzido à metade a cada 10 épocas sem redução no valor da função de perda. A otimização é terminada se o valor da função de perda não melhorar após 35 épocas.

$$
RMSE = \sqrt{\frac{1}{n_in_j}\sum_{i=1}^{n_i}\sum_{j=1}^{n_j}(y_{i,j} - \hat{y}(x_{i,j}))^2} \tag{5.1}
$$

6. Resultados

Foram feitos testes manuais com os hiperparâmetros da rede (Tabela 1). Dos testes realizados é possível tecer

Tabela 1: Hiperparâmetros do modelo *ótimo*.

Hiperparâmetro	**Valor**
Blocos intermediários	8
Canais no espaço latente:	64
Batch normalization	Sim
Ligação direta *externa*	Sim
Ligações diretas *internas*	Sim
Bloco Inception	Sim
Tamanho do *batch*	128

as seguintes conclusões:

- O incremento no número de blocos intermediários dificulta o treinamento da rede. As ligações diretas foram importantes para facilitar a *transmissão* do gradiente para as camadas mais rasas.
- O incremento no número de canais levou a melhora dos resultados. O contínuo incremento deixou o treinamento da rede instável. O aumento no tamanho do *batch* ajudou na redução das variações na função de perda ao longo do treinamento.
- O bloco Inception apresentou impacto positivo nos resultados. Comparando com uma camada convolucional de mesmo número de canais, o bloco Inception tem menos pesos ajustáveis e melhor desempeho.

Para aumentar a capacidade de generalização do modelo *ótimo*, o treinamento incluiu a variação da escala dos valores da matriz $\mathbf{A}$ e do vetorb. Os valores do vetor $\mathbf{y}$ foram mantidos na faixa [-2,2] para que a função de perda seja comparável entre as amostras. O treinamento atingiu o critério de parada após 304 épocas (Figura 4).

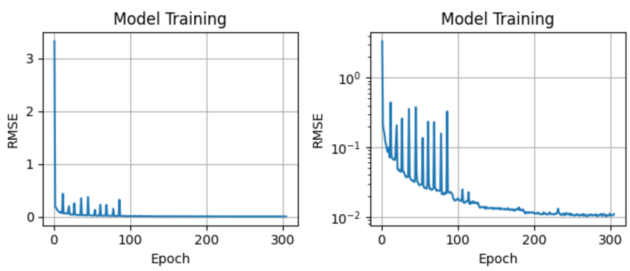

Figura 4: Treinamento do modelo *ótimo*.

O treinamento do modelo foi realizado com amostras de malhas $(5,5)$. Os resultados mostram que, mesmo que exista certa deteriorização da qualidade das respostas para malhas com outras configurações, o modelo conseguiu generalizar de modo satisfatório o problema proposto. As figuras de 5 a 9 mostram exemplos de aplicação do modelo treinado. Os vetores y_{exato} e y_{modelo} são apresentados como uma série apenas para facilitar a visualização.

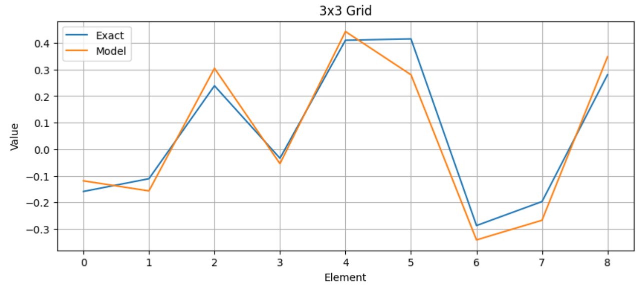

Figura 5: Exemplo de uma malha (3,3).

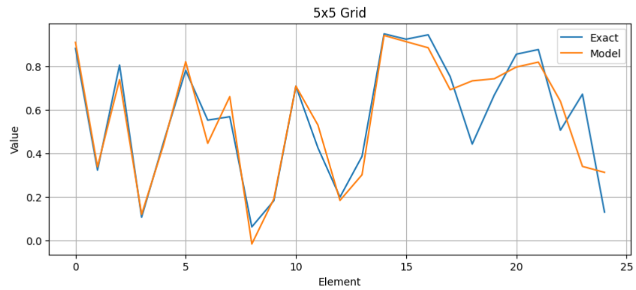

Figura 6: Exemplo de uma malha (5,5).

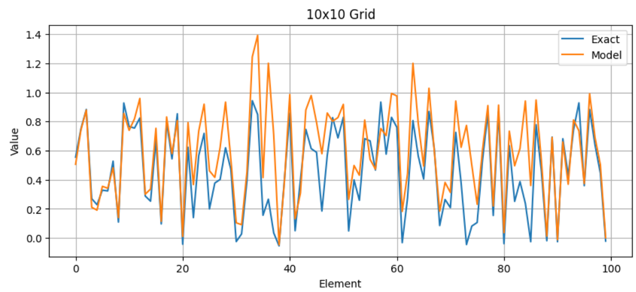

Figura 7: Exemplo de uma malha (10,10).

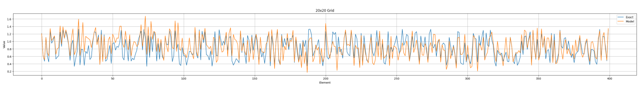

Figura 8: Exemplo de uma malha (20,20).

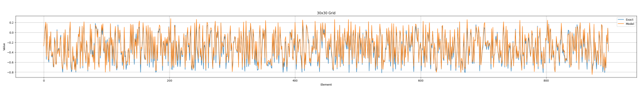

Figura 9: Exemplo de uma malha (30,30).

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. Conclusões

A proposta inicial para este projeto foi de resolver um passo de tempo de uma simulação de fluxo em meio poroso. Esta tarefa se mostrou desafiadora, pois o que se espera da rede neural é que *aprenda* a resolver um problema que da maneira tradicional envolve montar as matrizes de coeficientes a partir dos dados brutos do problema (porosidades, permeabilidades, viscosidades etc.) e resolver um sistema de equações não-lineares.

O objetivo deste trabalho não foi o de propor uma nova abordagem para a solução de sistemas pentadiagonais. Existem algoritmos de alta performance que tratam deste tipo de problema [8, 9, 10]. Neste trabalho buscou-se encontrar soluções para um problema mais simples que o inicialmente proposto, e aproveitar as lições aprendidas para a solução da proposta inicial.

O gráfico do treinamento do modelo *ótimo* aponta para uma saturação da função de perda, que pode ser indicativo de a falta de capacidade da própria rede neural de atingir melhores resultados. Testes com redes maiores tiveram problemas de convergência. Uma opção de trabalho futuro é treinar redes maiores a partir de redes menores já treinadas. Este mecanismo funciona como uma espécie de *transfer learning*, em que as novas camadas irão ajudar a melhorar a resposta das camadas já treinadas.

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.

[7] A. Zhang, Z. C. Lipton, M. Li, A. J. Smola, Dive into deep learning, arXiv preprint arXiv:2106.11342 (2021).

[8] C. Levit, Parallel solution of pentadiagonal systems using generalized odd-even elimination, in: Proceedings of the 1989 ACM/IEEE conference on Supercomputing, 1989, pp. 333–336.

[9] I. G. Ivanov, C. Walshaw, A parallel method for solving pentadiagonal systems of linear equations, Vol. 9, CMS Press, 1998.

[10] E. Carroll, A. Gloster, M. D. Bustamante, L. Ó. Náraigh, A batched gpu methodology for numerical solutions of partial differential equations, arXiv preprint arXiv:2107.05395 (2021).