Deep Galerkin Method

Resolvendo EDPs Numericamente com Redes Neurais

Caio Lins

FGV - EMAp

Novembro 2021

Sumário

Sumário

Formulação do problema

EDP com condições iniciais e de fronteira

Dados

- $\Omega \subseteq \mathbb{R}^d$ e $T > 0$ – domínio
- $u_0 : \Omega \to \mathbb{R}$ – condição inicial
- $g : [0, T] \times \partial\Omega \to \mathbb{R}$ – condição de fronteira,

Formulação do problema

EDP com condições iniciais e de fronteira

Dados

- $\Omega \subseteq \mathbb{R}^d$ e $T > 0$ – domínio
- $u_0 : \Omega \to \mathbb{R}$ – condição inicial
- $g : [0, T] \times \partial\Omega \to \mathbb{R}$ – condição de fronteira,

queremos encontrar $u : [0, T] \times \Omega \to \mathbb{R}$ tal que

$$\begin{cases} \partial_t u(t, \boldsymbol{x}) + \mathcal{L} u(t, \boldsymbol{x}) = 0, & (t, \boldsymbol{x}) \in [0, T] \times \Omega \\ u(0, \boldsymbol{x}) = u_0(\boldsymbol{x}), & \boldsymbol{x} \in \Omega \\ u(t, \boldsymbol{x}) = g(t, \boldsymbol{x}), & (t, \boldsymbol{x}) \in [0, T] \times \partial\Omega \end{cases}$$

Formulação do problema

Aqui $\mathcal{L}$ é um operador diferencial como, por exemplo, o laplaciano ∇^2:

$$\nabla^2 u = \frac{\partial^2 u}{\partial x_1^2} + \cdots + \frac{\partial^2 u}{\partial x_d^2}.$$

Métodos numéricos

- A solução numérica de EDPs é um desafio substancial.

Métodos numéricos

- A solução numérica de EDPs é um desafio substancial.
- Métodos numéricos non-network:

Métodos numéricos

- A solução numérica de EDPs é um desafio substancial.
- Métodos numéricos non-network:
  - *Métodos de diferenças finitas*
    Aproxima os valores da solução $u(t, \boldsymbol{x})$ em um *grid* utilizando uma equação de recorrência.

Métodos numéricos

- A solução numérica de EDPs é um desafio substancial.
- Métodos numéricos non-network:
  - *Métodos de diferenças finitas*
    Aproxima os valores da solução $u(t, \boldsymbol{x})$ em um *grid* utilizando uma equação de recorrência.
    Por exemplo, para a equação do calor $\partial_t u = \partial_{xx} u$:

$$u(t+\Delta t, \boldsymbol{x}) \approx u(t, \boldsymbol{x}) + \mu[u(t, \boldsymbol{x}+\Delta \boldsymbol{x}) - 2u(t, \boldsymbol{x}) + u(t, \boldsymbol{x}-\Delta \boldsymbol{x})].$$

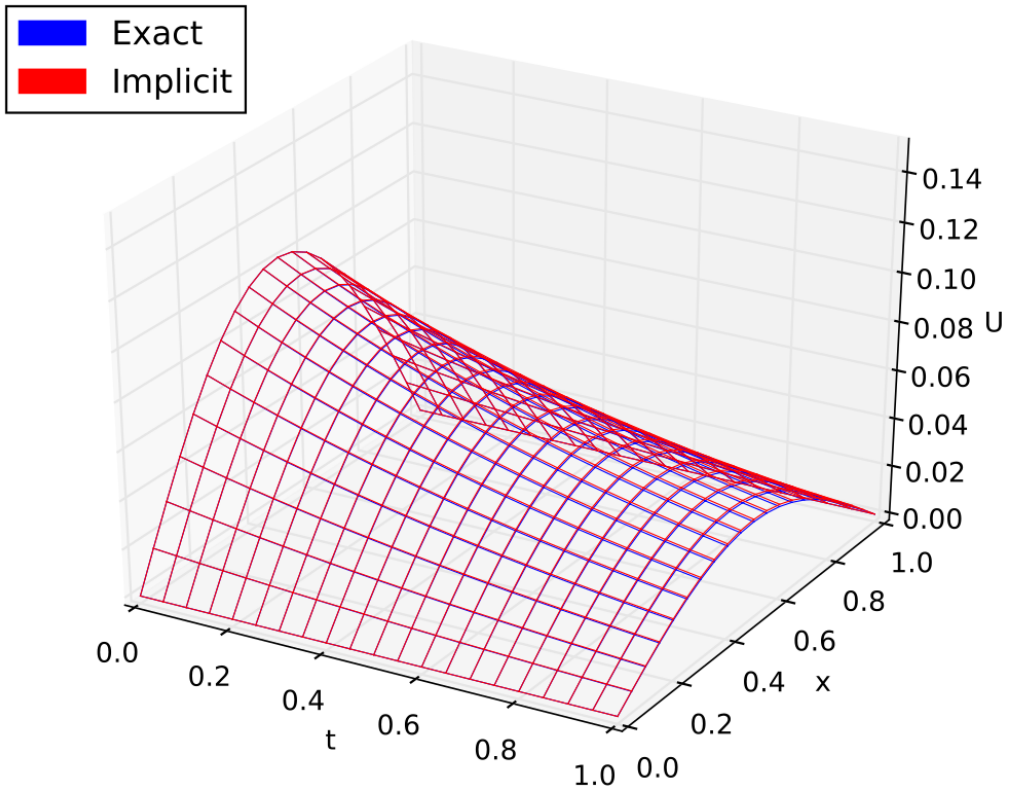
Métodos numéricos

- A solução numérica de EDPs é um desafio substancial.
- Métodos numéricos non-network:
  - *Métodos de diferenças finitas*
    Aproxima os valores da solução $u(t, \boldsymbol{x})$ em um *grid* utilizando uma equação de recorrência.
    Por exemplo, para a equação do calor $\partial_t u = \partial_{xx} u$:

    $$u(t+\Delta t, \boldsymbol{x}) \approx u(t, \boldsymbol{x}) + \mu[u(t, \boldsymbol{x}+\Delta \boldsymbol{x}) - 2u(t, \boldsymbol{x}) + u(t, \boldsymbol{x}-\Delta \boldsymbol{x})].$$

  - *Métodos de elementos finitos*
    Utiliza um espaço de funções de dimensão finita para aproximar a solução. Por exemplo, funções lineares ou polinomiais por partes.
    Também faz uso de um *grid*.

Métodos Numéricos



Exemplo de *grid* aplicado no método de diferenças finitas.

Métodos Numéricos

- Principal problema: a necessidade do *grid*.

Métodos Numéricos

- Principal problema: a necessidade do *grid*.
- Número de pontos no *grid* cresce exponencialmente com o número de dimensões espaciais.

Métodos Numéricos

- Principal problema: a necessidade do *grid*.
- Número de pontos no *grid* cresce exponencialmente com o número de dimensões espaciais.
- Métodos envolvendo Deep Learning passaram a contornar esse problema.

Sumário

Ideia Principal

- Vamos tentar aproximar a solução da EDP em $[0, T] \times \Omega$ com uma rede neural $f(t, \boldsymbol{x}, \theta)$

Ideia Principal

- Vamos tentar aproximar a solução da EDP em $[0, T] \times \Omega$ com uma rede neural $f(t, \boldsymbol{x}, \theta)$
  - Essa ideia faz sentido, pois redes neurais são aproximadores universais de funções contínuas [Hor91].

Ideia Principal

- Vamos tentar aproximar a solução da EDP em $[0, T] \times \Omega$ com uma rede neural $f(t, \boldsymbol{x}, \theta)$
  - Essa ideia faz sentido, pois redes neurais são aproximadores universais de funções contínuas [Hor91].
- Função de perda teórica, com três partes:

$$\begin{aligned} J(f) = &\int_{[0,T]\times\Omega} \|\partial_t f(t, \boldsymbol{x}, \theta) + \mathcal{L} f(t, \boldsymbol{x}, \theta)\|^2 \, \mathrm{d}\nu_1 \\ &+ \int_{[0,T]\times\partial\Omega} \|f(t, \boldsymbol{x}, \theta) - g(t, \boldsymbol{x})\|^2 \, \mathrm{d}\nu_2 \\ &+ \int_{\Omega} \|f(0, \boldsymbol{x}, \theta) - u_0(\boldsymbol{x})\|^2 \, \mathrm{d}\nu_3, \end{aligned}$$

onde ν_1, ν_2, ν_3 são medidas finitas.

Ideia Principal

- Na prática, para calcular a perda em um conjunto de pontos

$$\begin{cases} (t_i, \boldsymbol{x}_i) \subset [0,T] \times \Omega, \\ (\tau_i, \boldsymbol{y}_i) \subset [0,T] \times \partial\Omega, \\ \boldsymbol{w}_i \subset \Omega, \end{cases}$$

$i = 1, \ldots, n$, utilizamos o MSE:

$$\begin{aligned} G(f) =& \frac{1}{n} \sum_{i=1}^{n} |\partial_t f(t_i, \boldsymbol{x}_i, \theta) + \mathcal{L} f(t_i, \boldsymbol{x}_i, \theta)|^2 \\ &+ \frac{1}{n} \sum_{i=1}^{n} |f(\tau_i, \boldsymbol{y}_i, \theta) - g(\tau_i, \boldsymbol{y}_i)|^2 \\ &+ \frac{1}{n} \sum_{i=1}^{n} |f(0, \boldsymbol{w}_i, \theta) - u_0(\boldsymbol{w}_i)|^2. \end{aligned}$$

Primeiros Trabalhos

NEURAL-NETWORK-BASED APPROXIMATIONS FOR SOLVING PARTIAL DIFFERENTIAL EQUATIONS

M. W. M. G. DISSANAYAKE AND N. PHAN-THIEN
Department of Mechanical Engineering, The University of Sydney, Sydney, N.S.W. 2006, Australia

SUMMARY
A numerical method, based on neural-network-based functions, for solving partial differential equations is reported in the paper. Using a 'universal approximator' based on a neural network and point collocation, the numerical problem of solving the partial differential equation is transformed to an unconstrained minimization problem. The method is extremely easy to implement and is suitable for obtaining an approximate solution in a short period of time. The technique is illustrated with the aid of two numerical examples.

[DP94] utiliza essa ideia para solucionar equações bidimensionais definidas em $\Omega = [0, 1]^2$.

Primeiros Trabalhos

Características principais:

- Uso de NNs do tipo feedforward como aproximadores.

Primeiros Trabalhos

Características principais:

- Uso de NNs do tipo feedforward como aproximadores.
- Otimização feita utilizando um método quasi-Newton.

Primeiros Trabalhos

Características principais:

- Uso de NNs do tipo feedforward como aproximadores.
- Otimização feita utilizando um método quasi-Newton.
- Gradientes aproximados utilizando diferenças finitas.

Primeiros Trabalhos

Características principais:

- Uso de NNs do tipo feedforward como aproximadores.
- Otimização feita utilizando um método quasi-Newton.
- Gradientes aproximados utilizando diferenças finitas.
- Pontos de treino são tomados de um *grid* no domínio.

Primeiros Trabalhos

IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 9, NO. 5, SEPTEMBER 1998

Artificial Neural Networks for Solving Ordinary and Partial Differential Equations

Isaac Elias Lagaris, Aristidis Likas, *Member, IEEE,* and Dimitrios I. Fotiadis

***Abstract*—We present a method to solve initial and boundary value problems using artificial neural networks. A trial solution of the differential equation is written as a sum of two parts. The first part satisfies the initial/boundary conditions and contains no adjustable parameters. The second part is constructed so as not to affect the initial/boundary conditions. This part involves a feedforward neural network containing adjustable parameters (the weights). Hence by construction the initial/boundary conditions are satisfied and the network is trained to satisfy the differential equation. The applicability of this approach ranges from single ordinary differential equations (ODE's), to systems of coupled ODE's and also to partial differential equations (PDE's). In this article, we illustrate the method by solving a variety of model problems and present comparisons with solutions obtained using the Galekrkin finite element method for several cases of partial differential equations. With the advent of neuroprocessors and digital signal processors the method becomes particularly interesting due to the expected essential gains in the execution speed.**

of a feedforward neural network by replacing each spline with the sum of piecewise linear activation functions that correspond to the hidden units. This method considers local basis-functions and in general requires many splines (and consequently network parameters) in order to yield accurate solutions. Furthermore, it is not easy to extend these techniques to multidimensional domains.

In this article we view the problem from a different angle. We present a method for solving both ordinary differential equations (ODE's) and partial differential equations (PDE's) (defined on *orthogonal box* domains) that relies on the function approximation capabilities of feedforward neural networks and results in the construction of a solution written in a differentiable, closed analytic form. This form employs a feedforward neural network as the basic approximation element, whose parameters (weights and biases) are adjusted to

[LLF98] emprega esse método para solucionar EDOs, bem como EDPs definidas em cubos unitários.

Primeiros Trabalhos

Principais diferenças:

- Solução não é exclusivamente uma rede neural. Se Ψ_θ é a solução aproximada parametrizada por θ, temos

$$\Psi_\theta(\vec{\boldsymbol{x}}) = A(\vec{\boldsymbol{x}}) + F(\vec{\boldsymbol{x}}, f(\vec{\boldsymbol{x}}, \theta)),$$

onde $f(\vec{\boldsymbol{x}}, \theta)$ é a rede neural.

Primeiros Trabalhos

Principais diferenças:

- Solução não é exclusivamente uma rede neural. Se Ψ_θ é a solução aproximada parametrizada por θ, temos

$$\Psi_\theta(\vec{\boldsymbol{x}}) = A(\vec{\boldsymbol{x}}) + F(\vec{\boldsymbol{x}}, f(\vec{\boldsymbol{x}}, \theta)),$$

onde $f(\vec{\boldsymbol{x}}, \theta)$ é a rede neural.
A ideia é que $A(\vec{\boldsymbol{x}})$ não tenha parâmetros e satisfaça condições de fronteira. O termo $F(\vec{\boldsymbol{x}}, f(\vec{\boldsymbol{x}}, \theta))$ é encarregado fazer Ψ_θ obeder a EDP/EDO.

Primeiros Trabalhos

Principais diferenças:

- Por exemplo, considerando a EDO:

$$u'(x) = g(x, u),$$

com condição inicial $\Psi(0) = A$, a solução é escrita como

$$\Psi_\theta(x) = A + xf(x, \theta).$$

Primeiros Trabalhos

Principais diferenças:

- Por exemplo, considerando a EDO:

$$u'(x) = g(x, u),$$

com condição inicial $\Psi(0) = A$, a solução é escrita como

$$\Psi_\theta(x) = A + xf(x, \theta).$$

- Além disso, passa a utilizar diferenciação automática.

Primeiros Trabalhos

Entretanto:

- Rede utilizada ainda é uma MLP.

Primeiros Trabalhos

Entretanto:

- Rede utilizada ainda é uma MLP.
- Otimização continua quasi-Newton.

Primeiros Trabalhos

Entretanto:

- Rede utilizada ainda é uma MLP.
- Otimização continua quasi-Newton.
- *Grid* de treinamento se mantém.

Desenvolvimentos Mais Recentes

Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations

Maziar Raissi[1], Paris Perdikaris[2], and George Em Karniadakis[1]

[1]*Division of Applied Mathematics, Brown University, Providence, RI, 02912, USA*
[2]*Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA, 19104, USA*

Nov 2017

[RPK17] cunham o termo *Physics Informed Neural Network – PINN*

Desenvolvimentos Mais Recentes

- Uma PINN não necessariamente está relacionada a uma EDP.

Desenvolvimentos Mais Recentes

- Uma PINN não necessariamente está relacionada a uma EDP.
- A ideia é incorporar informações vindas da física no processo de treinamento e, assim, fazer um uso mais eficiente dos dados disponíveis.
  - Em sistemas físicos reais, o custo da aquisição de dados muitas vezes é demasiadamente elevado.

Desenvolvimentos Mais Recentes

- Uma PINN não necessariamente está relacionada a uma EDP.
- A ideia é incorporar informações vindas da física no processo de treinamento e, assim, fazer um uso mais eficiente dos dados disponíveis.
  - Em sistemas físicos reais, o custo da aquisição de dados muitas vezes é demasiadamente elevado.
- Entretanto, nos exemplos desse artigo especificamente, a ideia é incorporar a EDP na função de perda, como já descrevemos.

Desenvolvimentos Mais Recentes

Como agora o objetivo é ter a possibilidade de usar dados coletados, o conjunto de treino é um pouco diferente:

Desenvolvimentos Mais Recentes

Como agora o objetivo é ter a possibilidade de usar dados coletados, o conjunto de treino é um pouco diferente:

- Poucos pontos são tomados nas condições iniciais.
  - 100 pontos, por exemplo.

Desenvolvimentos Mais Recentes

Como agora o objetivo é ter a possibilidade de usar dados coletados, o conjunto de treino é um pouco diferente:

- Poucos pontos são tomados nas condições iniciais.
  - 100 pontos, por exemplo.
- A maior parte do conjunto de treino é tomada no interior de $[0, T] \times \Omega$.
  - E.g. 10000 pontos distribuídos aleatoriamente.

Desenvolvimentos Mais Recentes

Como agora o objetivo é ter a possibilidade de usar dados coletados, o conjunto de treino é um pouco diferente:

- Poucos pontos são tomados nas condições iniciais.
  - 100 pontos, por exemplo.
- A maior parte do conjunto de treino é tomada no interior de $[0, T] \times \Omega$.
  - E.g. 10000 pontos distribuídos aleatoriamente.
- Ou seja, não há mais um *grid*. Os pontos no interior do domínio são amostrados uma única vez antes de começar o treinamento.

Desenvolvimentos Mais Recentes

- Rede utilizada ainda é uma MLP, porém agora com mais camadas.

Desenvolvimentos Mais Recentes

- Rede utilizada ainda é uma MLP, porém agora com mais camadas.
- Fizeram uso da diferenciação automática do Tensorflow.

Desenvolvimentos Mais Recentes

- Rede utilizada ainda é uma MLP, porém agora com mais camadas.
- Fizeram uso da diferenciação automática do Tensorflow.
- O algoritmo de otimização ainda é quasi-Newton.

Sumário

Principais Características

DGM: A deep learning algorithm for solving partial differential equations

Justin Sirignano* and Konstantinos Spiliopoulos†‡§

September 7, 2018

Abstract

High-dimensional PDEs have been a longstanding computational challenge. We propose to solve high-dimensional PDEs by approximating the solution with a deep neural network which is trained to satisfy the differential operator, initial condition, and boundary conditions. Our algorithm is meshfree, which is key since meshes become infeasible in higher dimensions. Instead of forming a mesh, the neural network is trained on batches of randomly sampled time and space points. The algorithm is tested on a class of high-dimensional free boundary PDEs, which we are able to accurately solve in up to 200 dimensions. The algorithm is also tested on a high-dimensional Hamilton-Jacobi-Bellman PDE and Burgers' equation. The deep learning algorithm approximates the general solution to the Burgers' equation for a continuum of different boundary conditions and physical conditions (which can be viewed as a high-dimensional space). We call the algorithm a "Deep Galerkin Method (DGM)" since it is similar in spirit to Galerkin methods, with the solution approximated by a neural network instead of a linear combination of basis functions. In addition, we prove a theorem regarding the approximation power of neural networks for a class of quasilinear parabolic PDEs.

[SS18] Criaram um algoritmo que reúne as ideias até então apresentadas e faz algumas adições. Também provaram resultados teóricos sobre solucionar EDPs com redes neurais.

Principais Características

- *Arquitetura da rede*:
  A arquitetura utiliza um mecanismo de *gating* inspirado em redes do tipo LSTM/Highway [HS97] [SGS15].

Principais Características

- *Arquitetura da rede*:
  A arquitetura utiliza um mecanismo de *gating* inspirado em redes do tipo LSTM/Highway [HS97] [SGS15].
- *Conjunto de treino*:
  Para realizar **cada update nos parâmetros** são amostrados novos pontos no domínio da EDP (1000, por exemplo).
  O treino passa a ser contado em termos de *sampling stages* e não de épocas.

O Algoritmo

No n-ésimo *sampling stage*:

- Gerar um conjunto de pontos s_n pertencentes às três regiões de interesse: $[0, T] \times \Omega, [0, T] \times \partial\Omega$ e Ω.

O Algoritmo

No n-ésimo *sampling stage*:

- Gerar um conjunto de pontos s_n pertencentes às três regiões de interesse: $[0, T] \times \Omega, [0, T] \times \partial\Omega$ e Ω.
- Calcular o MSE: $G(\theta_n, s_n)$

O Algoritmo

No n-ésimo *sampling stage*:

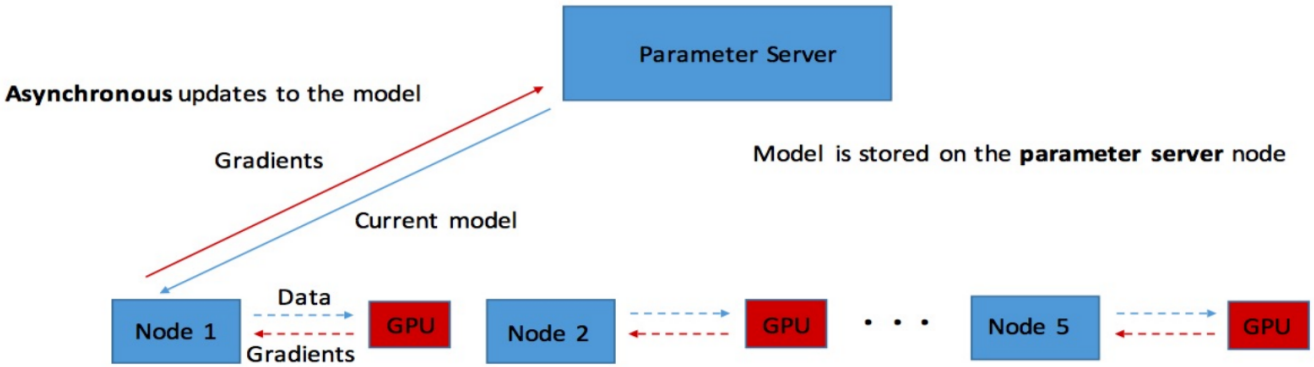
- Gerar um conjunto de pontos s_n pertencentes às três regiões de interesse: $[0,T] \times \Omega, [0,T] \times \partial\Omega$ e Ω.
- Calcular o MSE: $G(\theta_n, s_n)$
- Atualizar os parâmetros com o gradiente da perda:

$$\theta_{n+1} = \theta_n - \alpha_n \nabla_\theta G(\theta_n, s_n)$$

(na realidade, utilizamos o algoritmo ADAM).

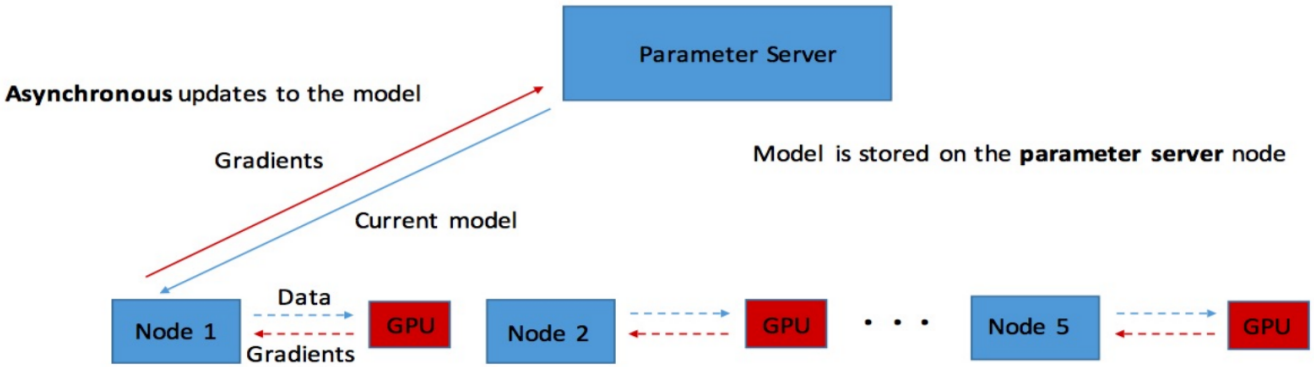
Outras Adições

- Treino **altamente paralelizado**:
  Realizado em 6 GPUs usando *asynchronous stochastic gradient descent* [Dea+12].



Outras Adições

- Treino **altamente paralelizado**:
  Realizado em 6 GPUs usando *asynchronous stochastic gradient descent* [Dea+12].



- Algoritmo Monte-Carlo para cálculo de segundas derivadas.
  Se torna mais rápido que diferenciação automática em muitas dimensões.

Arquitetura

- Camada inicial densa, seguida por L camadas do tipo Highway/LSTM-like, terminando com uma transformação afim (aqui, $\vec{\boldsymbol{x}} = (t, \boldsymbol{x})$):

$$
\begin{aligned}
S^1 &= \sigma(W^1 \vec{\boldsymbol{x}} + b^1) \\
Z^\ell &= \sigma\left(U^{z,\ell} \vec{\boldsymbol{x}} + W^{z,\ell} S^\ell + b^{z,\ell}\right), \ell = 1, \ldots, L, \\
G^\ell &= \sigma\left(U^{g,\ell} \vec{\boldsymbol{x}} + W^{g,\ell} S^\ell + b^{g,\ell}\right), \ell = 1, \ldots, L, \\
R^\ell &= \sigma\left(U^{r,\ell} \vec{\boldsymbol{x}} + W^{r,\ell} S^\ell + b^{r,\ell}\right), \ell = 1, \ldots, L, \\
H^\ell &= \sigma\left(U^{h,\ell} \vec{\boldsymbol{x}} + W^{h,\ell}\left(S^\ell \odot R^\ell\right) + b^{h,\ell}\right), \ell = 1, \ldots, L, \\
S^{\ell+1} &= \left(\mathbf{1} - G^\ell\right) \odot H^\ell + Z^\ell \odot S^\ell \\
f(t, \boldsymbol{x}, \theta) &= W S^{L+1} + b
\end{aligned}
$$

Arquitetura

- Temos $L+1$ camadas ocultas, cada uma com M nós.

Arquitetura

- Temos $L+1$ camadas ocultas, cada uma com M nós.
- A função $\sigma : \mathbb{R}^M \to \mathbb{R}^M$ é uma não-linearidade elemento a elemento:

$$\sigma(\boldsymbol{z}) = (\varphi(z_1), \varphi(z_2), \ldots, \varphi(z_M)),$$

onde $\varphi : \mathbb{R} \to \mathbb{R}$ é uma função de ativação não linear.

Arquitetura

- Temos $L+1$ camadas ocultas, cada uma com M nós.
- A função $\sigma : \mathbb{R}^M \to \mathbb{R}^M$ é uma não-linearidade elemento a elemento:

$$\sigma(\boldsymbol{z}) = (\varphi(z_1), \varphi(z_2), \ldots, \varphi(z_M)),$$

onde $\varphi : \mathbb{R} \to \mathbb{R}$ é uma função de ativação não linear.

- Parâmetros:

$$\theta = \Big\{ W^1, b^1, \left(U^{z,\ell}, W^{z,\ell}, b^{z,\ell}\right)_{\ell=1}^{L}, \left(U^{g,\ell}, W^{g,\ell}, b^{g,\ell}\right)_{\ell=1}^{L}, \left(U^{r,\ell}, W^{r,\ell}, b^{r,\ell}\right)_{\ell=1}^{L}, \left(U^{h,\ell}, W^{h,\ell}, b^{h,\ell}\right)_{\ell=1}^{L}, W, b \Big\}.$$

Arquitetura

Sugestões dos autores:

- $L = 3$.
- $M = 50$.
- $\varphi = \tanh$.

Avanços e Problemas

- Autores reportaram alta precisão na solução de equações com até 200 dimensões – ausência de *grid*.

Avanços e Problemas

- Autores reportaram alta precisão na solução de equações com até 200 dimensões – ausência de *grid*.
- Provaram um Teorema demonstrando a possibilidade de aproximar as soluções de uma classe de EDPs parabólicas quasilineares usando redes neurais.

Avanços e Problemas

Lembremos da função de perda teórica para uma rede $f(t, \boldsymbol{x}, \theta)$:

$$
\begin{aligned}
J(f) = &\int_{[0,T]\times\Omega} \|\partial_t f(t, \boldsymbol{x}, \theta) + \mathcal{L} f(t, \boldsymbol{x}, \theta)\|^2 \, \mathrm{d}\nu_1 \\
&+ \int_{[0,T]\times\partial\Omega} \|f(t, \boldsymbol{x}, \theta) - g(t, \boldsymbol{x})\|^2 \, \mathrm{d}\nu_2 \\
&+ \int_{\Omega} \|f(0, \boldsymbol{x}, \theta) - u_0(\boldsymbol{x})\|^2 \, \mathrm{d}\nu_3.
\end{aligned}
$$

Avanços e Problemas

Teorema

Seja $\mathfrak{C}^n$ *a classe de redes neurais com uma camada oculta e* n *nós. Seja* $f^n \in \mathfrak{C}^n$ *tal que* $J(f^n)$ *é mínimo em* $\mathfrak{C}^n$*. Sob certas condições, existe* $f^n \in \mathfrak{C}^n$ *tal que*

$$J(f^n) \to 0 \text{ quando } n \to \infty$$

e

$$f^n \to u \text{ quando } n \to \infty.$$

em $L^\rho([0,T] \times \Omega)$*, com* $\rho < 2$*.*

Avanços e Problemas

Principais questões pendentes:

- Dificuldade de implementação.

Avanços e Problemas

Principais questões pendentes:

- Dificuldade de implementação.
- Falta de garantias teóricas.

Referências I

[Hor91] K. Hornik. «Approximation Capabilities of Multilayer Feedforward Networks». Em: *Neural Networks* 4 (1991), pp. 251–257.

[DP94] M. W. M. G. Dissanayake e N. Phan-Thien. «Neural-Network-Based Approximations for Solving Partial Differential Equations». Em: *Communications in Numerical Methods in Engineering* 10 (1994), pp. 195–201.

[HS97] S. Hochreiter e J. Schmidhuber. «Long Short-Term Memory». Em: *Neural Computation* 9 (1997), pp. 1735–1780.

Referências II

[LLF98] I. E. Lagaris, A. Likas e D. I. Fotiadis. «Artificial Neural Networks for Solving Ordinary and Partial Differential Equations». Em: *IEEE Transactions on Neural Networks* 9 (1998), pp. 987–1000.

[Dea+12] J. Dean et al. «Large Scale Distributed Deep Networks». Em: *Advances in Neural Information Processing Systems* (2012), pp. 1223–1231.

[SGS15] R. K. Srivastava, K. Greff e J. Schmidhuber. «Highway Networks». Em: *arXiv preprint arXiv:1505.00387v2* (2015).

Referências III

[RPK17] Maziar Raissi, Paris Perdikaris e George Em Karniadakis. «Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations». Em: *arXiv preprint arXiv:1711.10561* (2017).

[SS18] J. Sirignano e K. Spiliopoulos. «DGM: A Deep Learning Algorithm for Solving Partial Differential Equations». Em: *arXiv preprint arXiv:1708.07469v5* (2018).