Solução de Equações Diferenciais Parciais com Redes Neurais Artificiais

Gilmar Francisco de O. Santos

FCT UNESP

gilmar.francisco@unesp.br

August 6, 2020

Sumário

Redes Neurais Artificiais

Redes neurais artificiais (RNAs) são modelos computacionais inspirados pelo sistema nervoso central de um animal (em particular o cérebro) que são capazes de realizar o aprendizado de máquina bem como o reconhecimento de padrões. Redes neurais artificiais geralmente são apresentadas como sistemas de "neurônios interconectados, que podem computar valores de entradas", simulando o comportamento de redes neurais biológicas.

Feedforward Neural Network

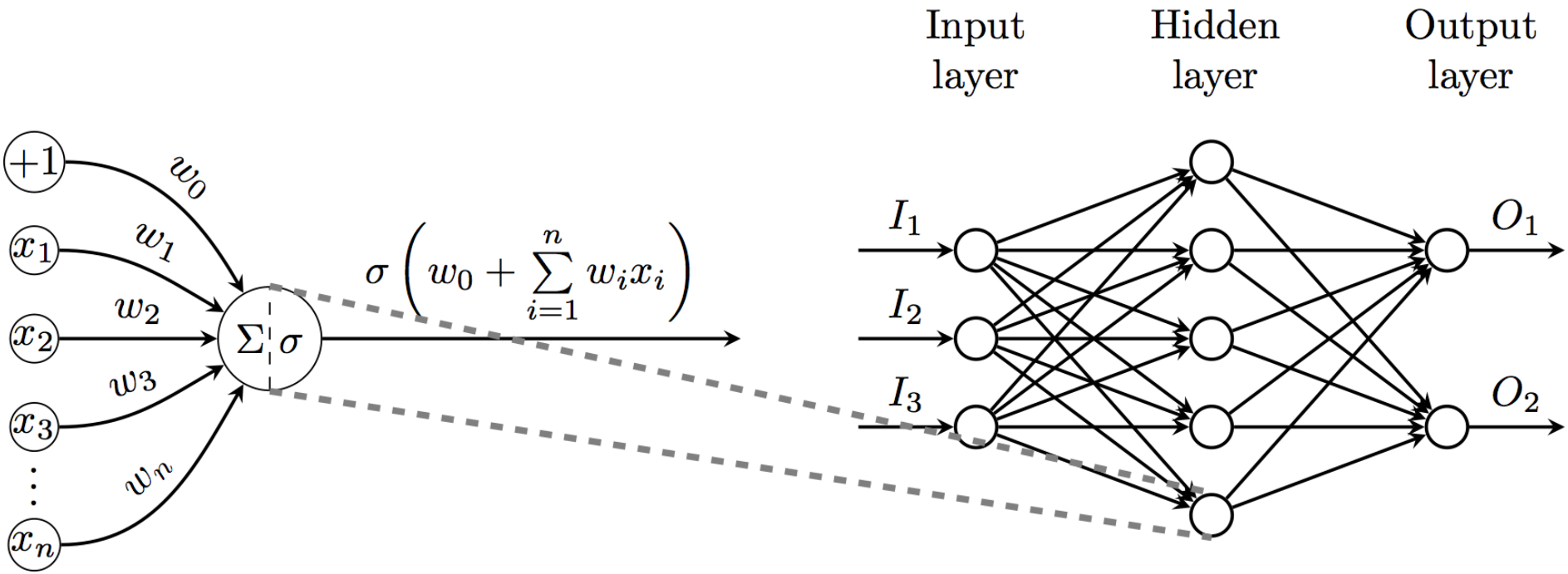


Figure: RNA com uma camada oculta

Funções de Ativação Típicas

- Sigmoid

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

- Tanh

$$\sigma(z) = \frac{e^{(z)} - e^{(-z)}}{e^{(z)} - e^{(-z)}}$$

- ReLu

$$\sigma(z) = max(z, 0)$$

Funções de Ativação Típicas

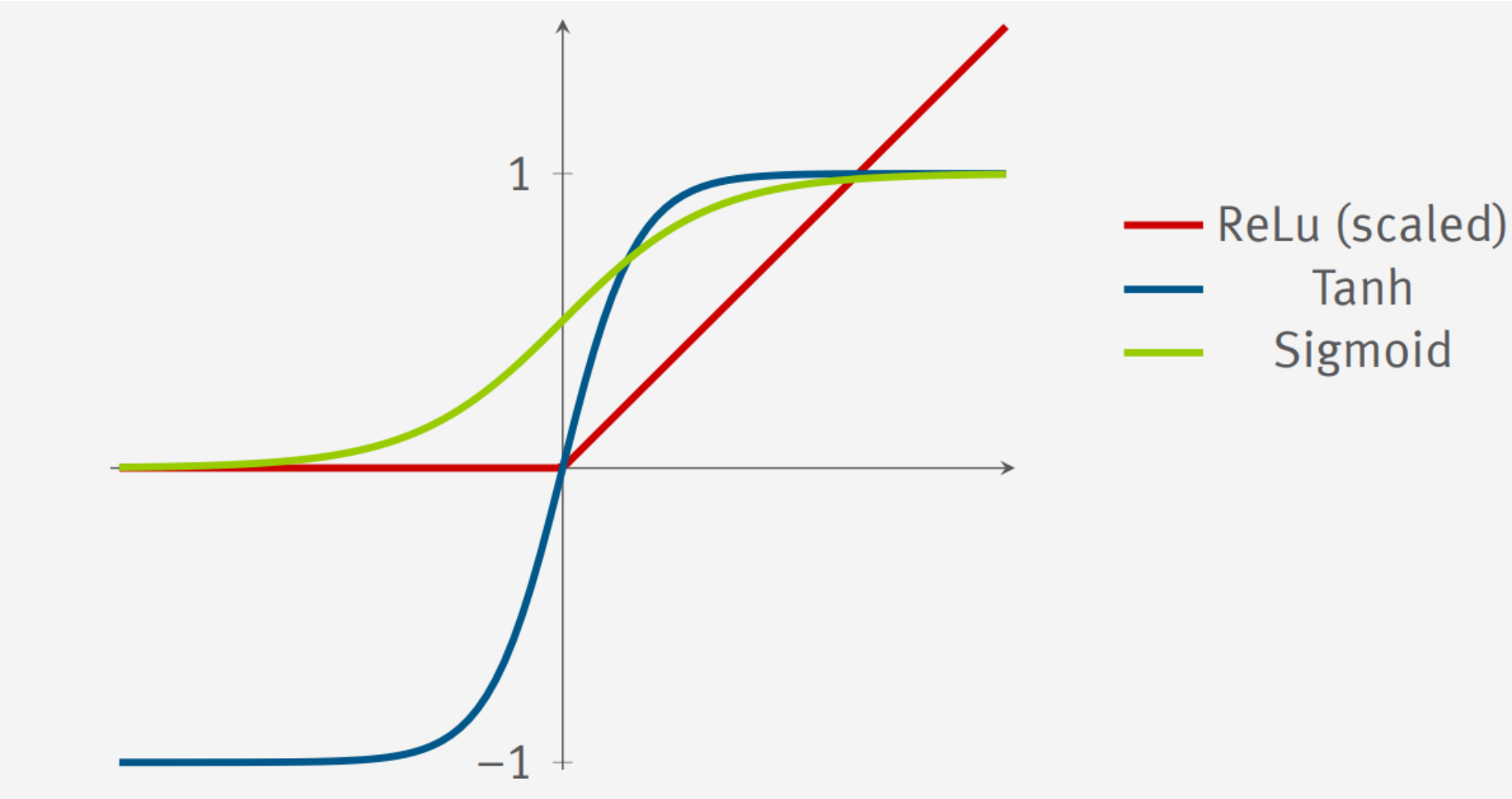


Figure: Funções de Ativação

Feedforward Neural Network (Formalização)

Dado um vetor de entrada (input) $z^l \in R^n$, a saída (output) de uma camada totalmente conectada $l+1$ é

$$z^{l+1} = \sigma \left(\sum_{i=1}^{n} W_{i,j}^l z_i^l + b_i^l \right)_{j=1,\ldots,m}, \quad \in R^m \tag{1}$$

Essas camadas podem ser concatenadas, formando uma Deep Neural Network, parametrizada pela matriz de pesos W^l e vetor de bias b^l para cada layer. Nós denotamos esses parâmetros por θ e a RNA correspondente por $N(\theta, \vec{x})$.

Treinamento

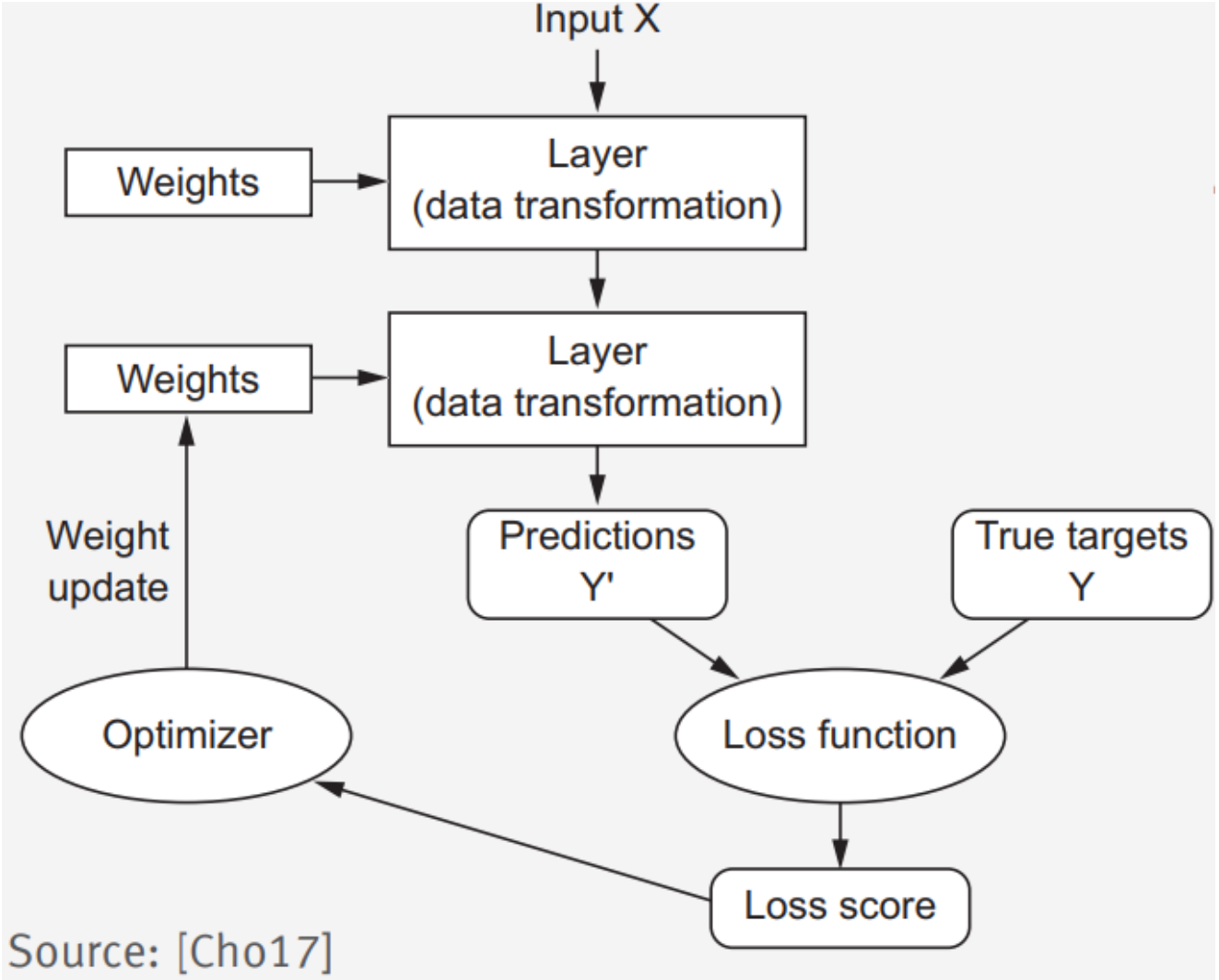


Figure: Loop de treinamento.

Teorema da Aproximação Universal

Na teoria matemática de RNAs, o teorema da aproximação universal declara que uma rede neural pré-alimentada com uma única camada oculta que contém um número finito de neurônios pode aproximar funções contínuas em subconjuntos compactos de R^n, com pressupostos mínimos de função de ativação. O teorema afirma que redes neurais simples (shallow networks) podem representar uma grande variedade de funções interessantes quando há os parâmetros adequados.
Kurt Hornik mostrou em 1991 que não é a escolha específica da função de ativação, mas sim o feedforward própria arquitetura de múltiplas camadas que dá redes neurais o potencial de ser aproximadores universais .
As unidades de saída são sempre assumido como sendo linear. Por conveniência de notação, apenas o caso de saída único será mostrado. O caso geral pode ser facilmente deduzido a partir do caso de saída única.

Teorema da Aproximação Universal

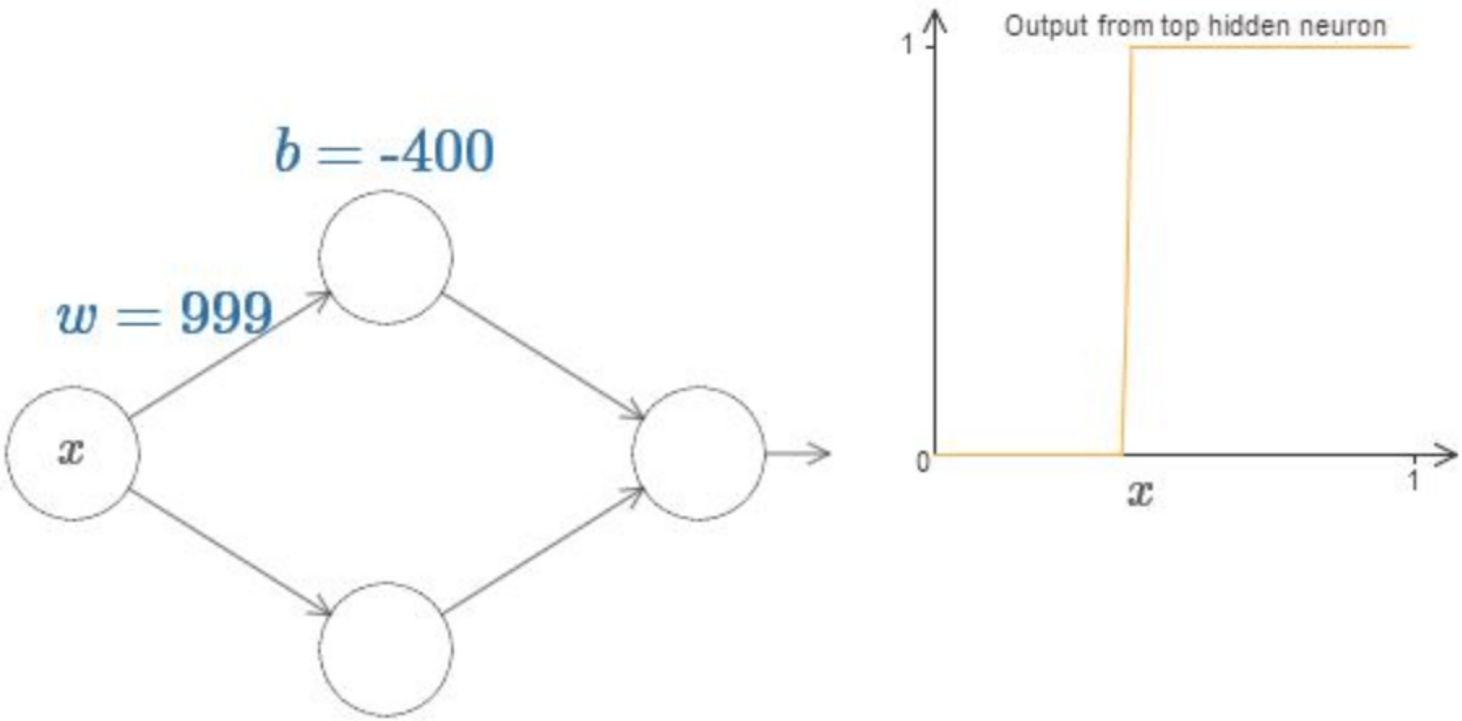


Figure: Aproximação usando de um neurônio

Teorema da Aproximação Universal

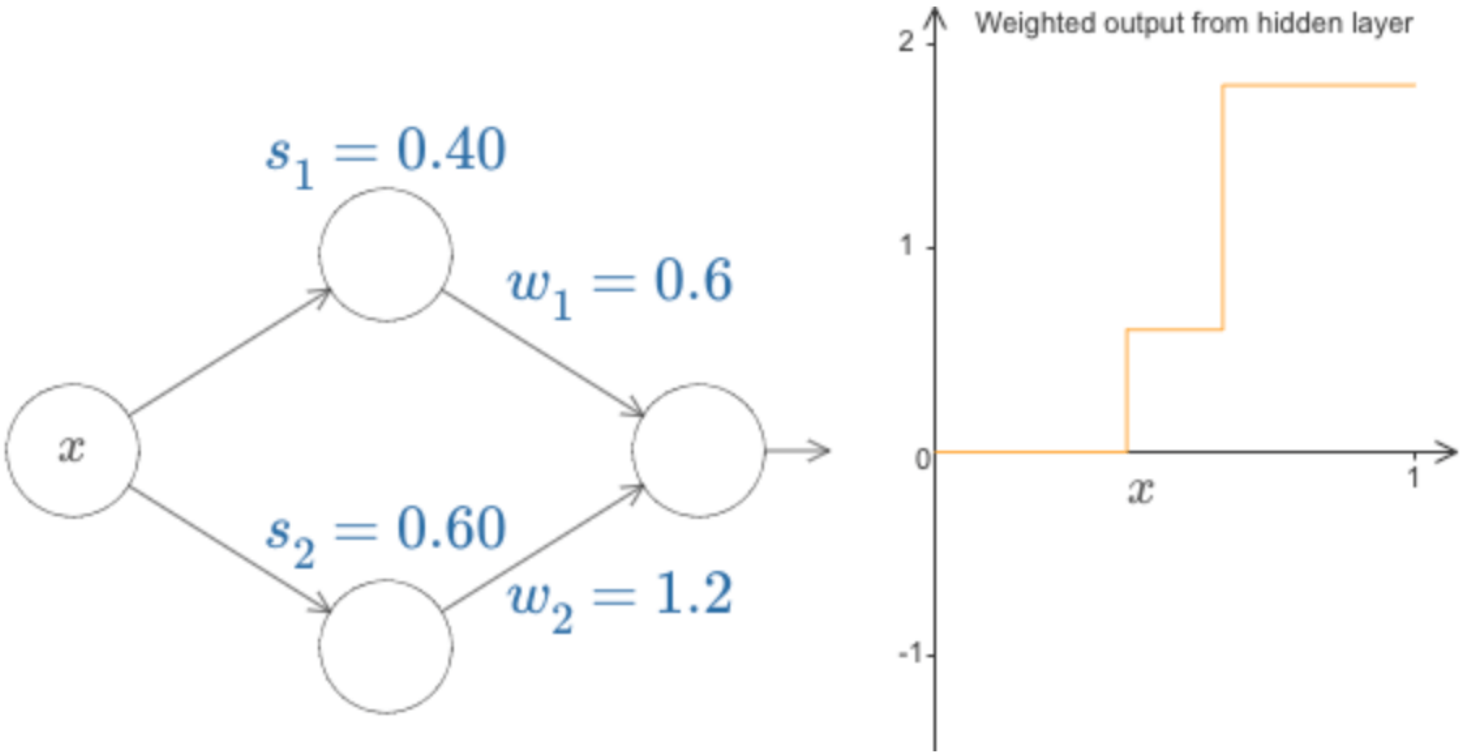


Figure: Realizando a composição de Funções

Teorema da Aproximação Universal

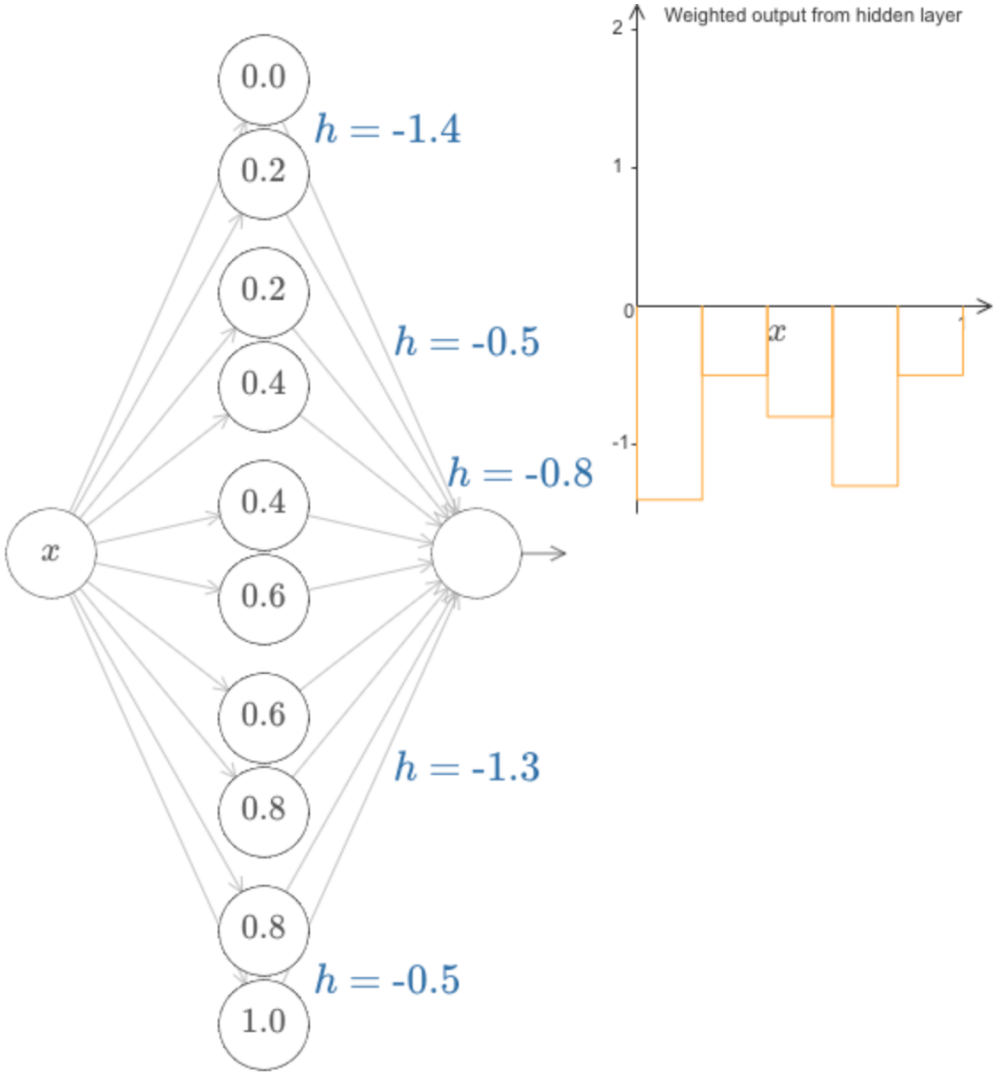


Figure: Aproximação completa

Gradient Descent

O Gradiente Descendent é um algoritmo de otimização usado para encontrar os valores dos parâmetros (coeficientes) de uma função f que minimiza a Função de Custo (Cost Function). O Gradiente Descendente é melhor empregado quando os parâmetros não podem ser calculados analiticamente e precisam ser buscados por um algoritmo de otmização.
Uma posição aleatória é o custo dos valores atuais dos coeficientes, queremos então minimizar este custo, encontrando o mínimo global para a função. Para isso continuamos testando diferentes valores para os coeficiente, avaliamos o custo para eles e então selecionamos coeficiente que são um pouco melhores que os anteriores.
Repetindo esse processo tempo suficiente, seremos levados a obter o menor custo e a encontrar os coeficientes que geram esse menor custo.

Gradient Descent

Sabe-se do Cálculo que o gradiente de uma função diferenciável aponta para sua direção de maior crescimento e é perpendicular a suas curvas de nível. Usando a função de erro com parâmetros $\theta = (w, b)$ o método do gradiente descendente permite a busca por pontos mínimos de uma maneira computacional usando a regra de iteração:

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta E(\theta | x, y) \tag{2}$$

onde α, conhecido como taxa de aprendizado, define o tamanho do passo em direção ao ponto de mínimo.

Gradient Descent

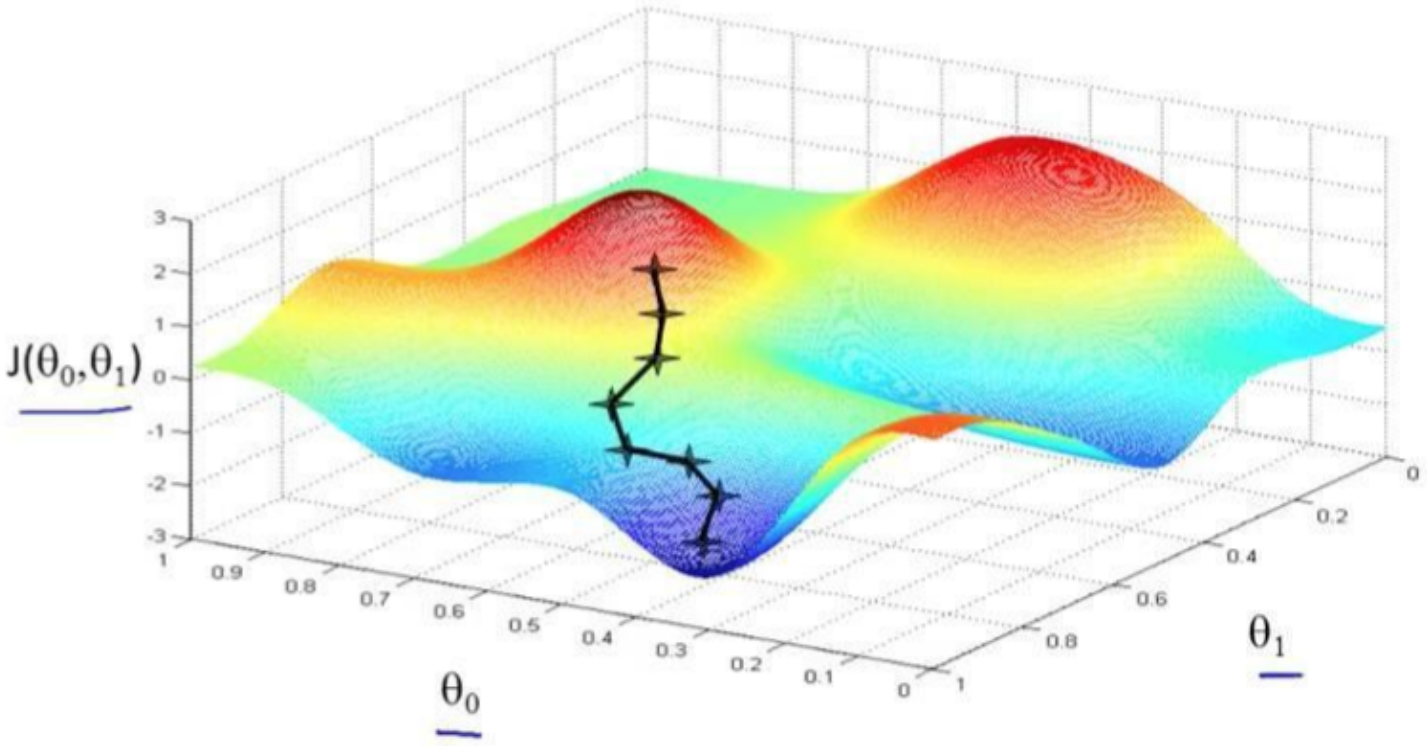


Figure: Processo de Minimização

Gradient Descent

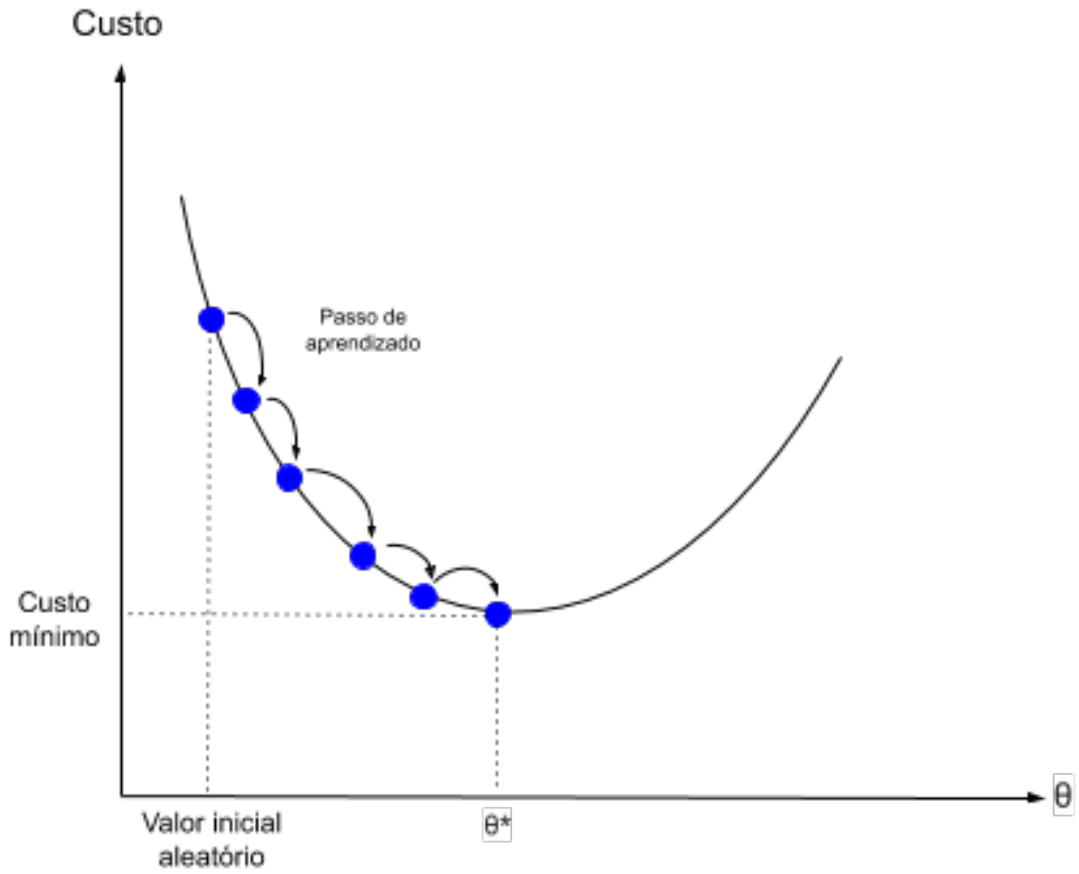


Figure: Influência do Learning Rate (α) no aprendizado

Diferenciação Automática

A diferenciação automática (DA), também chamada de diferenciação algorítmica, é um conjunto de técnicas para avaliar a derivada de função numéricas especificadas em um programa de computador.
A diferenciação automática explora o fato que todo programa computacional executa uma sequência de operações aritméticas (adição, subtração, multiplicação, divisão, etc.) e funções elementares (exponencial, logaritmo, seno, cosseno, etc.). Ao aplicar repetidamente a regra da cadeia a estas operações a derivada pode ser computada automaticamente e de modo eficiente, utilizando apenas algumas operações aritméticas a mais que a função original.

RNA para Solução de EDPs

A ideia de resolver equações diferenciais usando de Machine Learning foi introduzida em 1996 por Isaac Lagaris, Aristidis Likas e Dimitrios Fotiadis (University of Ioannina, Greece).
Essa abordagem tem as seguinte vantagens:

- A solução aprendida pela rede neural tem é diferenciável e de forma analítica fechada
- O método é geral
- Implementação eficiente em arquiteturas paralelas
- A solução aprendida pela RNA tem boa generalização em pontos não treinados
- Uma acurácia alta pode ser alcançada com poucos parâmetros.
- Não apresenta risco de overfitting

RNA para Solução de EDPs

Definimos que a rede $u_t = N(w, b)$ é uma aproximação para u, onde u é a solução para o problema de valor de contorno:

Theorem (Problema)

Definimos $\Omega \subseteq \mathbb{R}^n$. Dada uma equação diferencial geral da forma:

$$G(\vec{x}, \Psi(\vec{x}), \nabla\Psi(\vec{x}), \nabla^2\Psi(\vec{x})) = 0 \quad \in \Omega,$$

sujeita a condição de contorno: $\Psi(\vec{x}) = g(\vec{x}) \in \Gamma \subseteq \partial\Omega$, onde $\Psi : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}$, $G : \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^{n\times n} \to \mathbb{R}$ e $g : \Gamma \to \mathbb{R}$. Encontre a solução Ψ.

Parametrizamos então Ψ com uma rede neural, com $\Psi = \Psi_\theta$ para os parâmetros θ.

Definição da Função de Custo

Definimos a função de custo como sendo o resíduo de Ψ_θ aproximado pela RNA aplicado em G:

$$C(\Psi_\theta, \vec{x}) = G \tag{3}$$

Onde C representa a função de custo (resíduo), nosso objetivo é que esse valor seja o mais próximo possível de zero. Porém como vem da teoria, para que o PVF tenha uma solução única devemos adicionar condições de contorno ao mesmo.

Definição da Função de Loss

Definimos a função de Loss como sendo o resíduo entre o Ψ_θ aproximado pela RNA e as derivadas de grau inferior aplicadas na função f:

$$E(\theta) = \frac{1}{N}\left(\sum_{i=1}^{N} C(N(\theta, \vec{x_i}), \vec{x_i})^2\right)_{(\Omega)} \tag{4}$$

Onde E representa a função de loss (perda), nosso objetivo novamente é que esse valor seja o mais próximo possível de zero. E será justamente este valor como visto anteriormente que usaremos no Gradient Descendent.

Definição da Arquitetura da Rede

Como foi discutido, uma rede com apenas uma camada oculta (hidden layer) podemos aproximar qualquer função, desde que as condições para função de ativação e suficiente treinamento seja realizado.
Porém o Teorema da Aproximação Universal pode ser estendido de forma análoga para RNAs mais profundas, ou seja, com mais camadas ocultas. O maior número de camadas geralmente influencia positivamente na aproximação, dado que precisaremos logaritmicamente de um menor número de neurônios para aproximar as curvas da função, o que leva a um menor custo computacional para realizar o treinamento.

Lidando com as condições de contorno

Como assegurar que Ψ_θ satisfaça as condições de contorno? Na literatura observamos duas abordagens:

- Adicionar um termo de penalidade a loss function ("soft assignment")
- Usar trial solutions que satisfaçam automaticamente as condições de contorno ("hard assignment")

$$\Psi_t(\vec{x}) = A(\vec{x}) + F(\vec{x}, N_\theta(\vec{x}))$$

A função A é definida de modo a satisfazer as condições de contorno, F não contribui para as condições de contorno (define uma "distância" em relação aos contornos) e N_θ é uma rede neural de parâmetros θ. A e F precisam ser definidas "na mão" dependendo do problema a ser trabalhado. O que se torna algo não trivial para contornos complexos.

Hard assignment: Função para satisfazer o contorno

- Para contornos do tipo Dirichlet:

$$
\begin{aligned}
A(x,y) &= (1-x)f_0(y) + xf_1(y) + \\
&+ (1-y)\{g_0(x) - [(1-x)g_0(0) + xg_0(1)]\} + \\
&+ y\{g_1(x) - [(1-x)g_1(0) + xg_1(1)]\}
\end{aligned}
$$

- Para contornos mistos:

$$
\begin{aligned}
B(x,y) &= (1-x)f_0(y) + xf_1(y) + \\
&+ g_0(x) - [(1-x)g_0(0) + xg_0(1)] \\
&+ y\{g_1(x) - [(1-x)g_1(0) + xg_1(1)]\}
\end{aligned}
$$

Soft assignment: Penalidade no contorno

Modificamos nossa loss function de modo a "penalizar" a rede caso os valores de contorno não sejam satisfeitos. Para isso adicionamos os valores de contorno ficando então com o seguinte loss:

$$E(\theta) = \frac{1}{N}\left(\sum_{i=1}^{N} C(N(\theta, \vec{x_i}), \vec{x_i})^2\right)_{(\Omega)} + \sum [\Psi_\theta - g]^2_{(\Gamma)} \tag{5}$$

Onde $N(\theta, \vec{x_i}) = \Psi_\theta(\vec{x})$.

Considerações sobre RNAs e Malhas Computacionais

Diferentemente de Métodos Numéricos Clássicos, as RNAs recebem um impacto muito mais relevante da definição de seus parâmetros e hiperparâmetros, sendo assim, para muitos casos como a malha computacional é definida, não tem relevância primária, esse fator leva a uma aplicação útil das RNAs, para casos em que o domínio a ser trabalhado é muito complexo, sendo impossível e/ou muito custoso gerar uma malha "ótima" para ser usada no Método Numérico.
De um modo geral podemos simplesmente trabalhar com uma malha uniformemente espaçada e não sendo necessário um refinamento muito grande.

Exemplo 1: EDP Poisson - Trab. Prático 3-4 (Soft assignment)

Considerando o Problema 2 do Trabalho Prático:

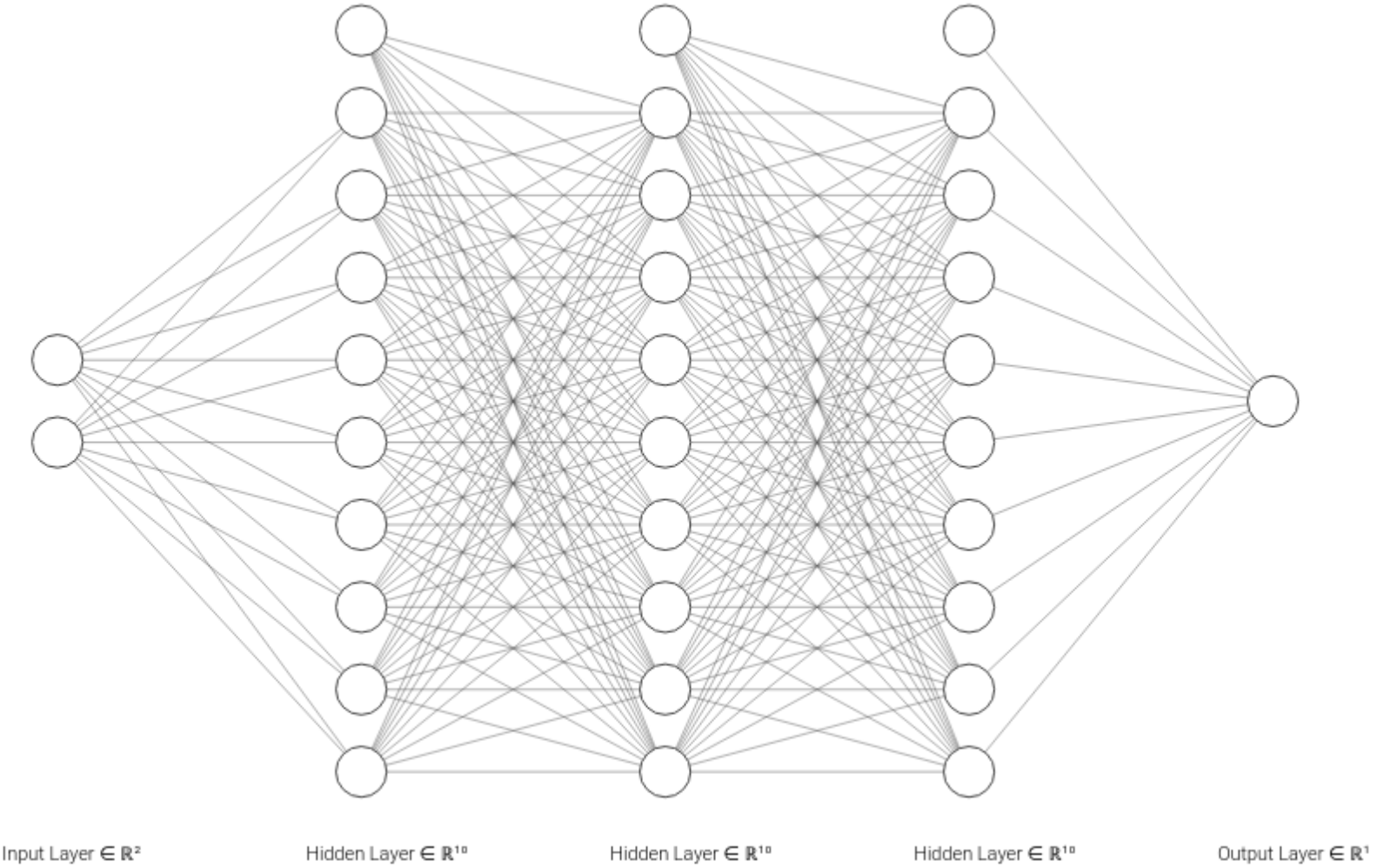
$$\frac{\partial^2 \Psi}{\partial x^2} + \frac{\partial^2 \Psi}{\partial y^2} = -2$$

Onde $\Omega = \{(x, y) \in R^2 : -1 \leq x \leq 1, -1 \leq y \leq 1\}$, sujeito as condições de contorno de Dirichlet $u = 0$ para $u \in \partial\Omega$.
Construímos a função de loss:

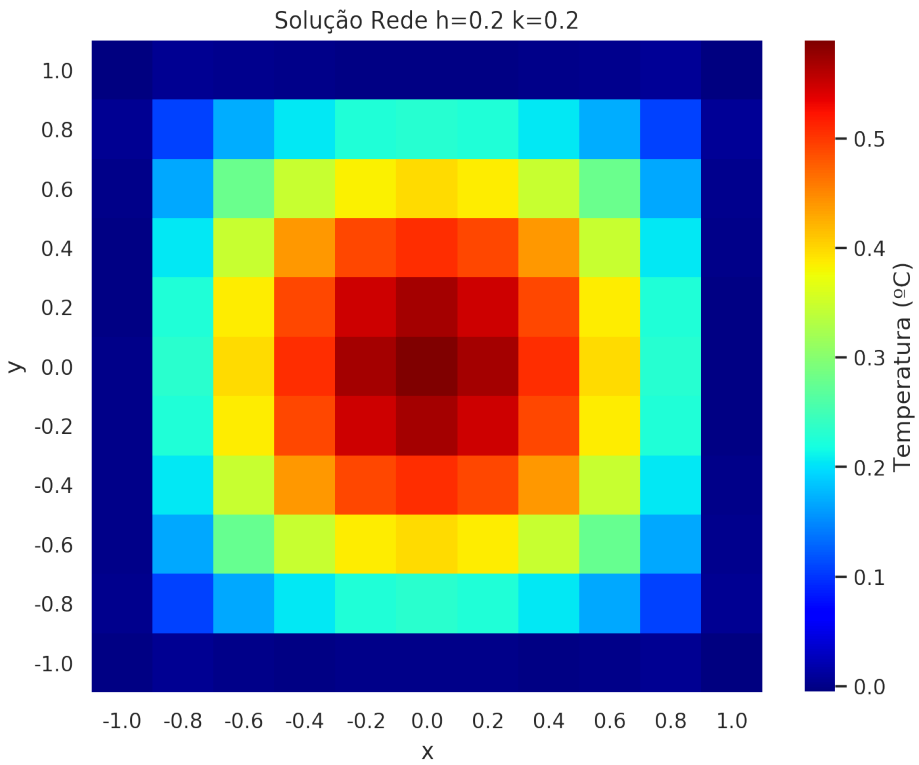
$$E(\theta) = \frac{1}{N * M} \sum_{i=1}^{N} \sum_{j=1}^{M} \left(\frac{\partial^2 \Psi_\theta(x_i, y_j)}{\partial x^2} + \frac{\partial^2 \Psi_\theta(x_i, y_j)}{\partial y^2} + 2 \right)^2 +$$

$$+ \sum_{i=1}^{N} \Psi_\theta(x_i, -1)^2 + \sum_{i=1}^{N} \Psi_\theta(x_i, 1)^2 + \sum_{j=1}^{M} \Psi_\theta(-1, y_j)^2 + \sum_{j=1}^{M} \Psi_\theta(-1, y_j)^2$$

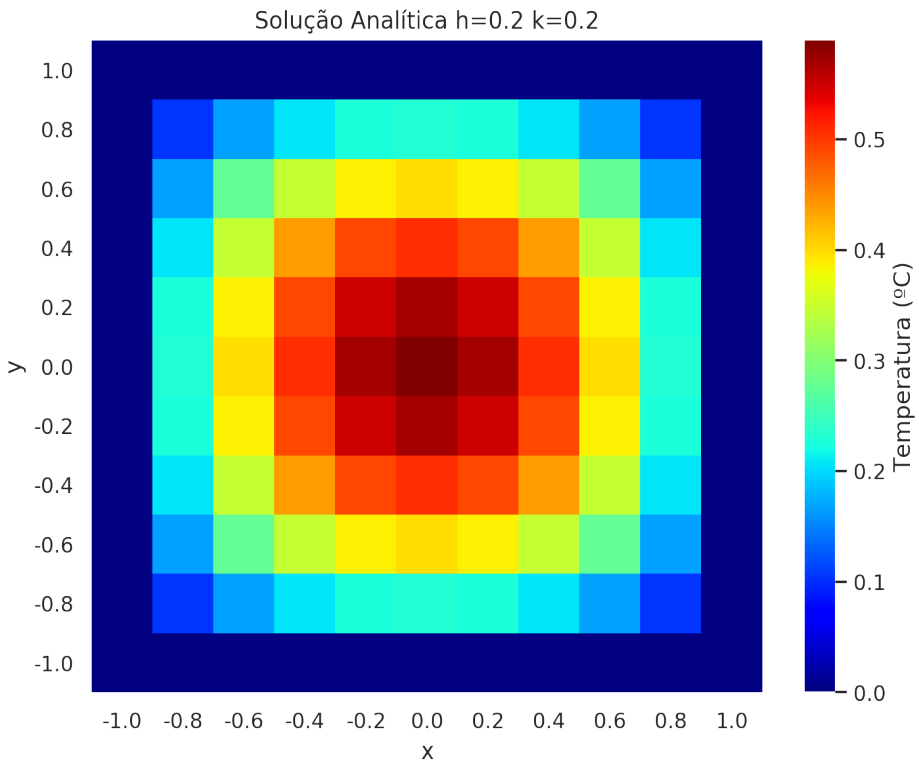
Exemplo 1: Arquitetura da RNA



Exemplo 1: Resultado

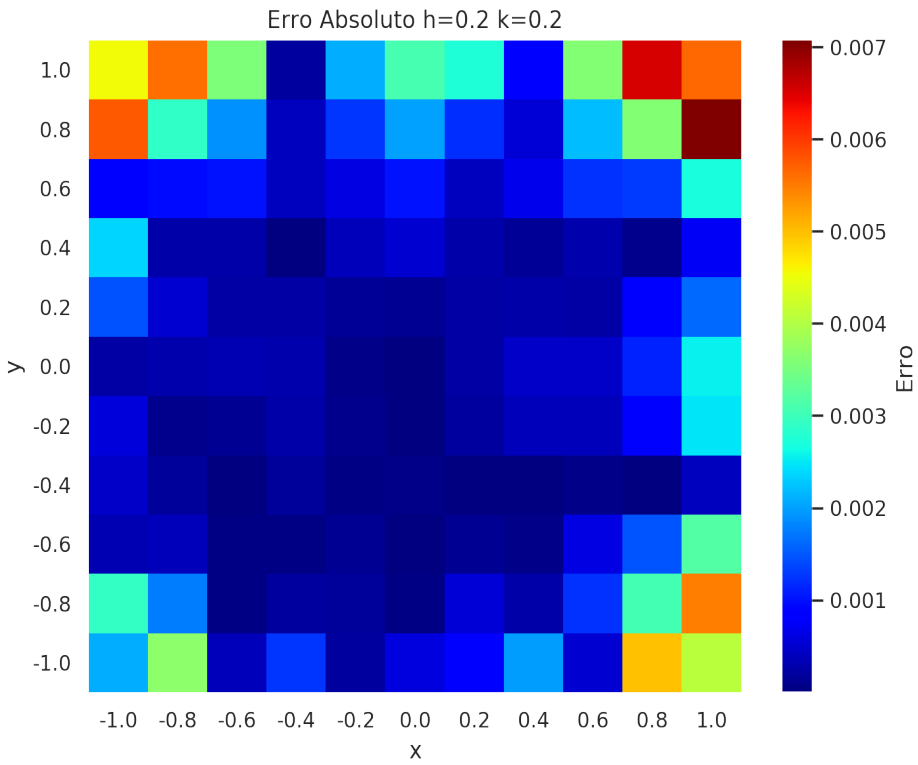


(a) Solução da Rede

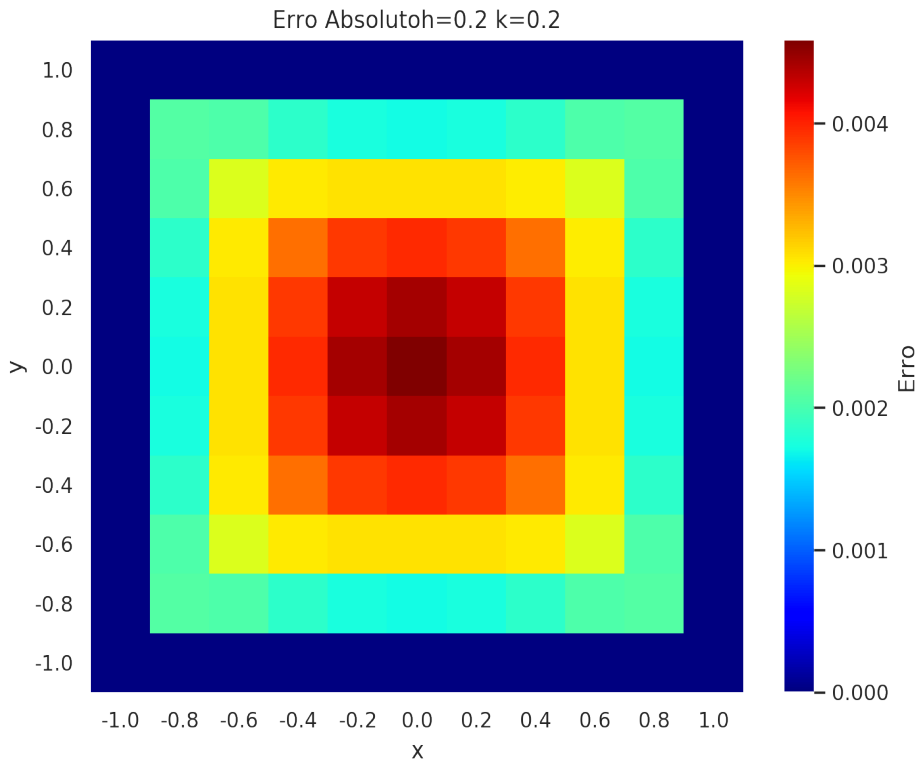


(b) Solução Analítica

Exemplo 1: Erro Absoluto Rede x Gauss-Seidel h=k=0.2

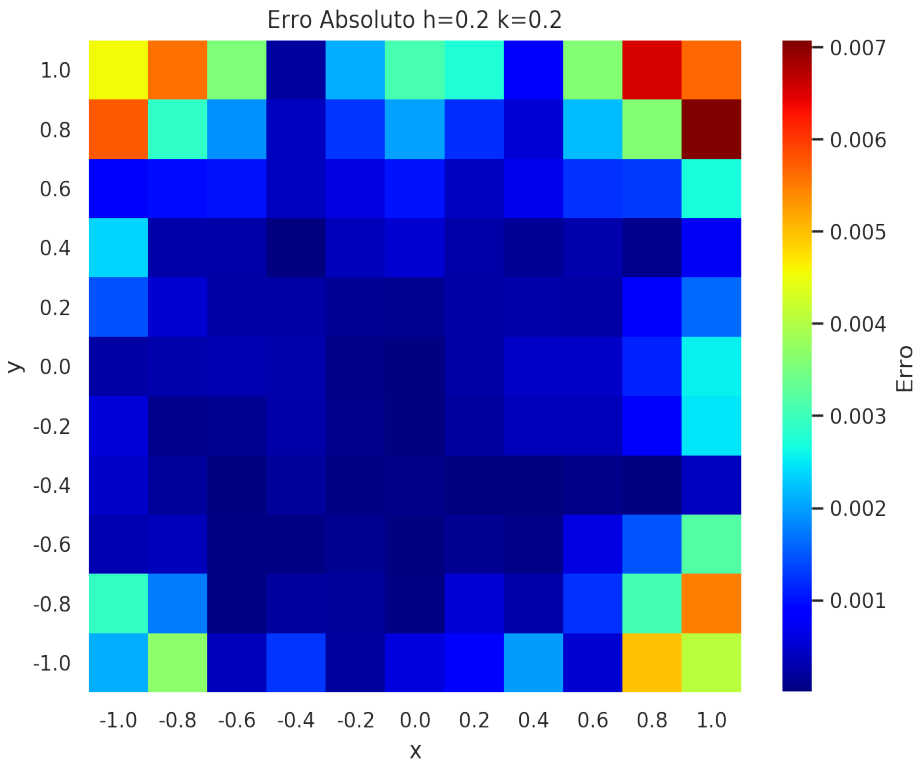


(c) Erro Absoluto Rede

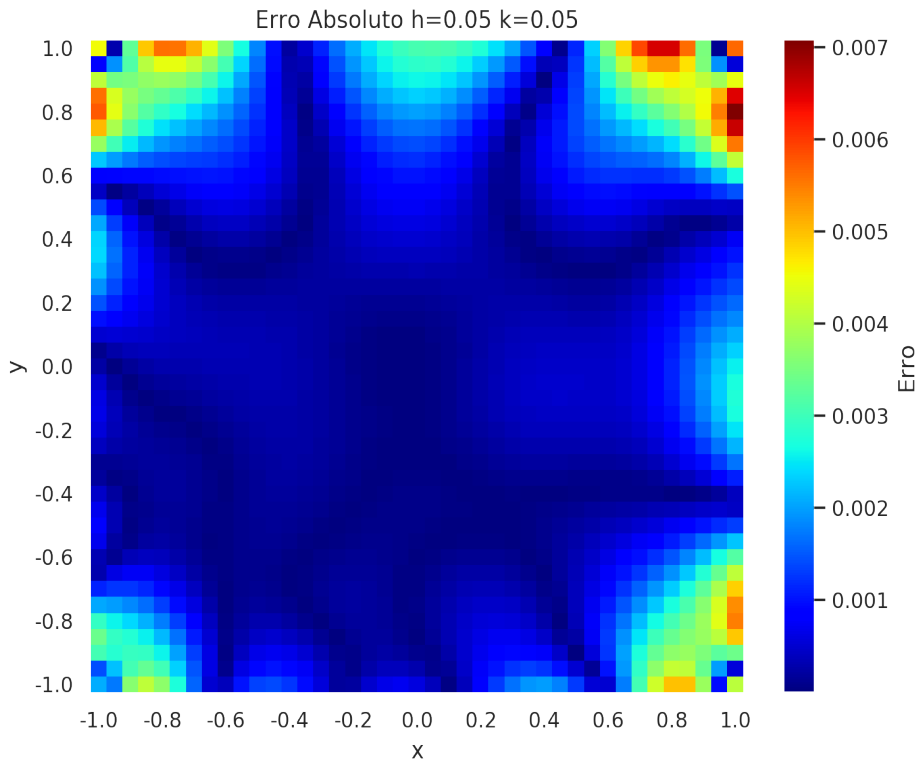


(d) Erro Absoluto Gauss-Seidel

Exemplo 1: Erro Absoluto Rede



(e) Erro Absoluto h=k=0.2



(f) Erro Absoluto h=k=0.05

Erro em Pontos Interpolados [Lagaris, 2000]

	Neural Method		*Finite Element*	
Problem No.	Training set	Interpolation set	Training set	Interpolation set
5	5×10^{-7}	5×10^{-7}	2×10^{-8}	1.5×10^{-5}
6	0.0015	0.0015	0.0002	0.0025
7	6×10^{-6}	6×10^{-6}	7×10^{-7}	4×10^{-5}
8	1.5×10^{-5}	1.5×10^{-5}	6×10^{-7}	4×10^{-5}

Figure: Comparativo erro em pontos interpolados Rede x FEM

Exemplo 2: Problema 7 [Lagaris, 1998]

Dado:

$$\frac{\partial^2 \Psi}{\partial x^2} + \frac{\partial^2 \Psi}{\partial y^2} = (2 - \pi^2 y^2) sin(\pi x)$$

Tomemos agora um problema com condições de contorno mistas. $\Psi(0, y) = \Psi(1, y) = \Psi(x, 0) = 0$ e $\frac{\partial \Psi(x,1)}{\partial y} = 2sin(\pi x)$. Onde $\Omega = \{(x, y) \in R^2 : 0 \leq x \leq 1, 0 \leq y \leq 1\}$.
Usando hard assignment para gerar Ψ_t ficamos com:

$$\Psi_t(x, y) = B(x, y) + x(1 - x)y[N_\theta(x, y) - N_\theta(x, 1) - \frac{\partial N_\theta(x, 1)}{\partial y}]$$

A função para satisfazer os contornos fica:

$$B(x, y) = y * (g_1(x) - ((1 - x) * g_1(0) + x * g_1(1)))$$

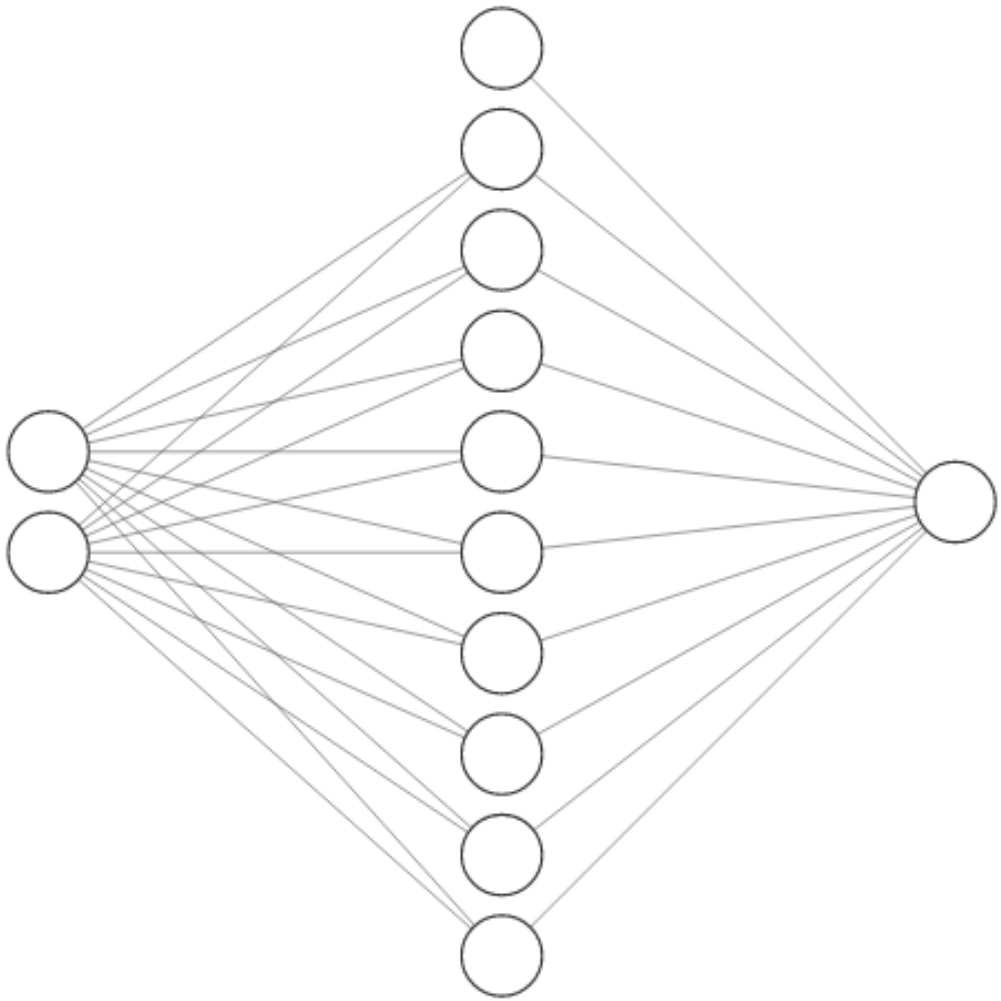
Onde $g_1 = 2sin(\pi x)$.

Exemplo 2: Problema 7 [Lagaris, 1998]

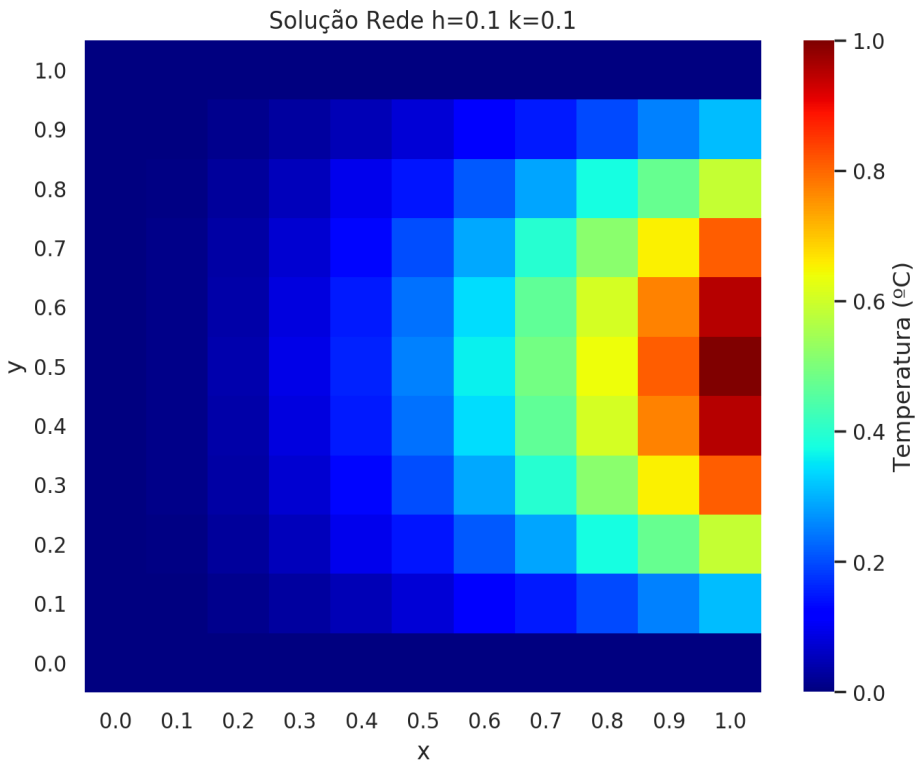
Construímos a função de loss:

$$E(\theta) = \sum_{i=1}^{N}\sum_{j=1}^{M}\left(\frac{\partial^2\Psi_\theta(x_i,y_j)}{\partial x^2} + \frac{\partial^2\Psi_\theta(x_i,y_j)}{\partial y^2} - (2-\pi^2 y_j^2)sin(\pi x_i)\right)^2$$

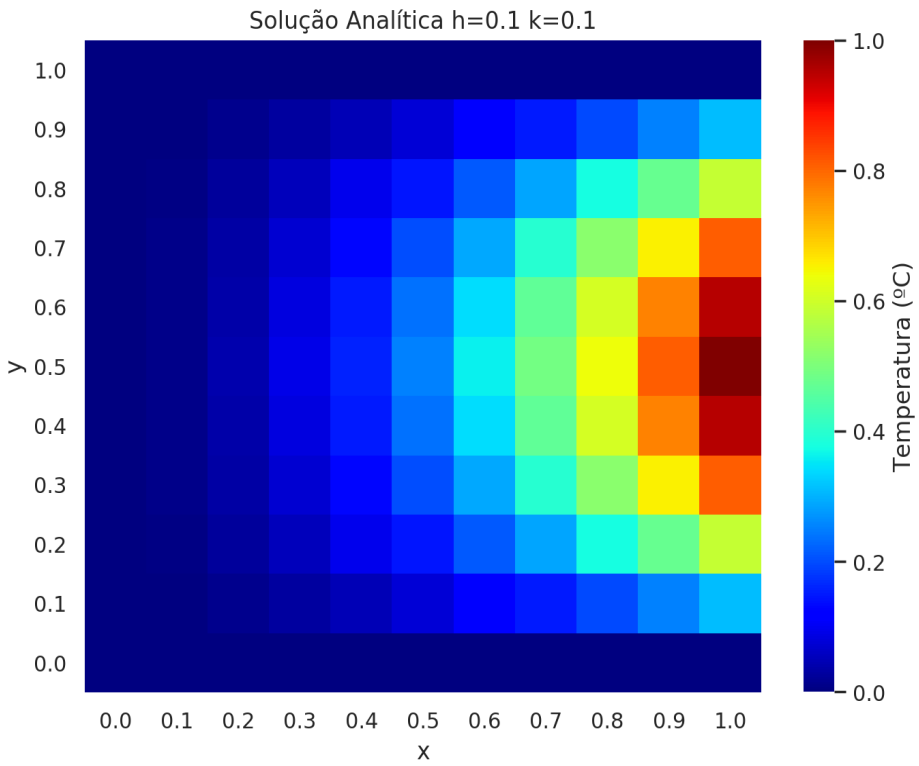
Exemplo 2: Arquitetura da RNA



Exemplo 2: Problema 7 [Lagaris, 1998]



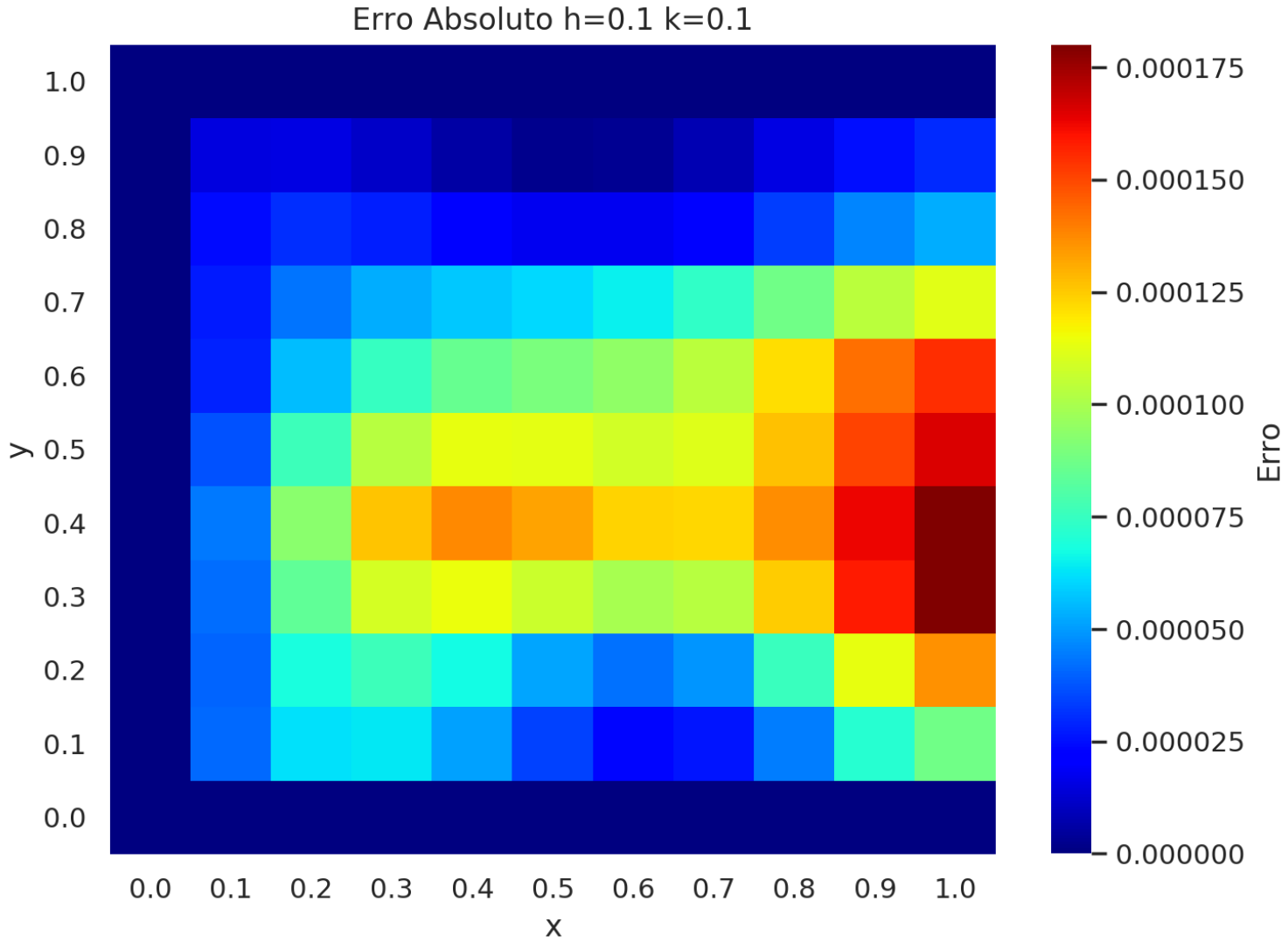
(a) Solução da Rede



(b) Solução Analítica

Exemplo 2: Erro Absoluto Rede

O erro segue o esperado, já que o contorno $\Psi_t(x, 1)$ foi o que apresentou maior erro.



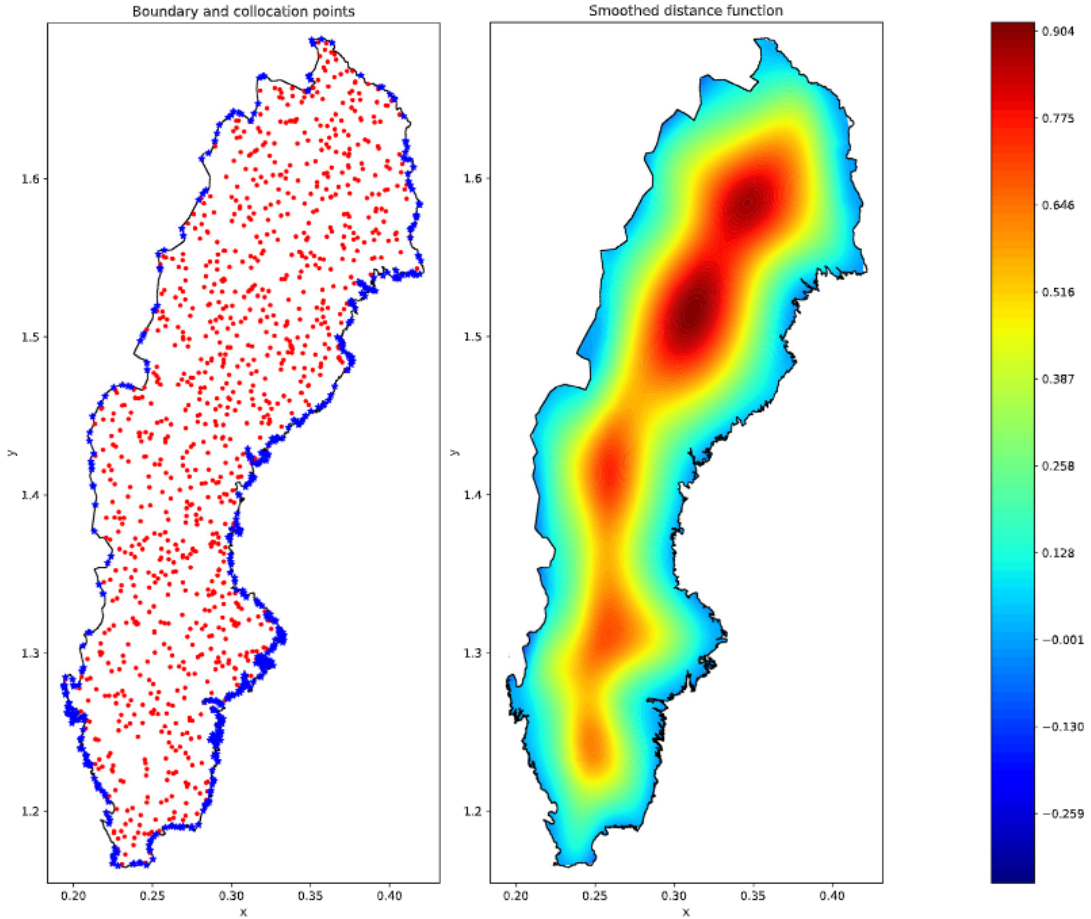
Exemplo 3: Domínio Mapa Suécia

Para domínios poligonais malhas de alta qualidade podem ser geradas, para esses casos a solução obtida pela RNA não é muito competitiva, porém considerando um domínio mais complexo, no caso o mapa da Suécia.

- Os autores relataram que uma malha ótima não pode ser gerada mesmo depois de 16h de processamento com um gerador de malha do pacote FEniCS (estado da arte no publicação do artigo)
- Domínio representado por um polígono com 160.876 vértices
- Treinar a RNA em um laptop demorou 10 minutos, ainda tendo margem para o desenvolvimento em bibliotecas como PyTorch ou TensorFlow.

Exemplo 3: Domínio Mapa Suécia

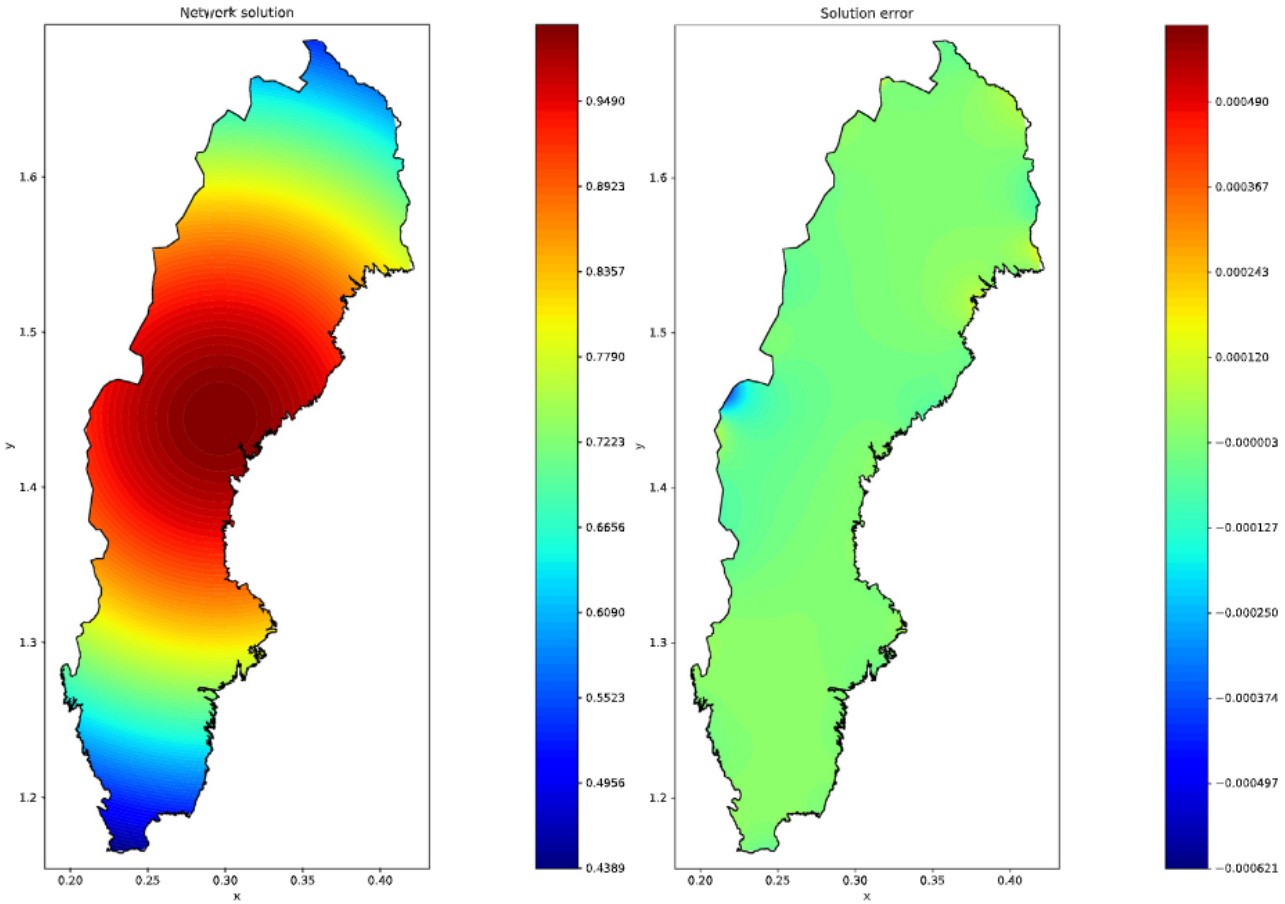
Resolver a equação de difusão linear em 2D. Cuja solução analítica é: $u(x, y) = e^{-10(x-m_x)^2+(y-m_y)^2}$ onde (m_x, m_y) é o centro de massa do domínio.



(a) Collocation and boundary points used to compute $d(x)$.

(b) Smoothed distance function using a single hidden layer with 20 neurons.

Exemplo 3: Domínio Mapa Suécia



(a) ANN solution to the 2D stationary diffusion equation.

(b) Difference between the exact and computed solution.

Figure: Solução pela Rede Neural com 5 camadas ocultas e 10 neurônios cada.

Influência do número de camadas na acurácia

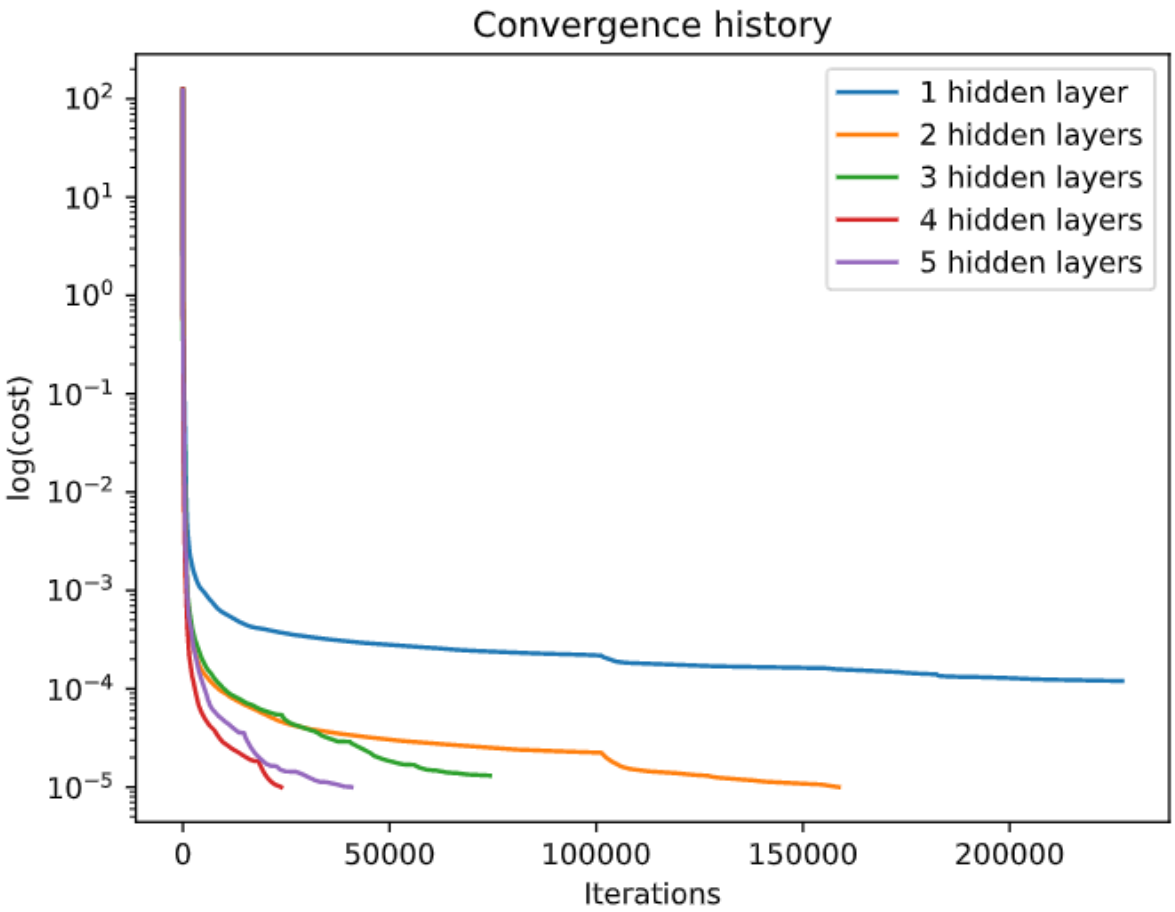


Figure: De um modo geral 4 camadas com 10 neurônios cada, se adequam melhor a problemas 2D.

Vantagens

- A RNA treinada pode atuar de mesmo modo que a solução analítica podendo ser derivada e aplicada em pontos não treinados.
- A construção da loss function é direta, então adaptação para cada tipo de problema tem um esforço mínimo
- O método é geral e independente da malha
- O número de pontos pode ser muito elevado, mas podem ser processados sequencialmente sem danos na convergência
- transfer learning pode ser utilizado para problemas similares
- Ampla disponibilidade de Frameworks para paralelização

Desvantagens

- A escolha da arquitetura da rede pode ser difícil
- Apresenta grande dependência da inicialização
- Dado que envolve geralmente um problema de minimização não convexo, pode ficar preso em mínimos locais.
- Não apresenta nenhuma análise de velocidade de convergência ou estabilidade.

Referências

- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.
- Lagaris, Isaac E., Aristidis Likas, and Dimitrios I. Fotiadis. "Artificial neural networks for solving ordinary and partial differential equations." IEEE transactions on neural networks 9.5 (1998): 987-1000.
- Lagaris, Isaac E., Aristidis C. Likas, and Dimitris G. Papageorgiou. "Neural-network methods for boundary value problems with irregular boundaries." IEEE Transactions on Neural Networks 11.5 (2000): 1041-1049.
- Berg, Jens, and Kaj Nyström. "A unified deep artificial neural network approach to partial differential equations in complex geometries." Neurocomputing 317 (2018): 28-41.
- Cybenko, George. "Approximation by superpositions of a sigmoidal function." Mathematics of control, signals and systems 2.4 (1989): 303-314.
- Csáji, Balázs Csanád. "Approximation with artificial neural networks." Faculty of Sciences, Etvs Lornd University, Hungary 24.48 (2001): 7.