Galerkin convida Mr. Deep para um café

Felipe Rocha
felipe.figueredorocha@epfl.ch
EAMC, LNCC, Petrópolis, 2021

February 9, 2021

Que título estranho!?

Metodos Numéricos :
Elementos Finitos,
Volumes Finitos,
Diferenças Finitas, etc

Boris Galerkin
1871-1945

Que título estranho!?

Metodos Numéricos :
Elementos Finitos,
Volumes Finitos,
Diferenças Finitas, etc

Boris Galerkin
1871-1945

Aprendizagem de Máquina:
Redes Neurais (Profundas),
Processos Gaussianos, K-means,
Máquinas de Vetores Suporte, etc

Mr. Deep
2000 - ?

Que título estranho!?

Metodos Numéricos :
Elementos Finitos,
Volumes Finitos,
Diferenças Finitas, etc

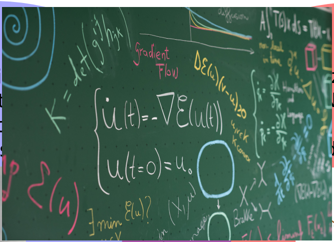
Boris Galerkin
1871-1945

Aprendizagem de Máquina:
Redes Neurais (Profundas),
Processos Gaussianos, K-means,
Máquinas de Vetores Suporte, etc

Mr. Deep
2000 - ?

Que título estranho!?

Metodos Numéri
Elementos Finit
Volumes Finito
Diferenças Finitas



zagem de Máquina:
leurais (Profundas),
Gaussianos, K-means,
de Vetores Suporte, etc

Boris Galerkin
1871-1945

Mr. Deep
2000 - ?

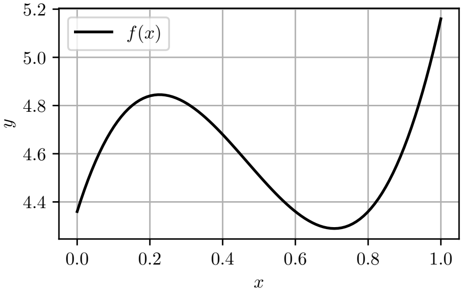
Objetivos do Workshop

- Fornecer um ponto de partida para aprender:
  - Elementos Finitos para quem é originalmente da área de Aprendizado de Máquina.
  - Redes Neurais (com foco Aprendizado Profundo) para quem é originalmente da área de Métodos Numéricos. Mais particularmente, sob a óptica da matemática aplicada.
- "Pré-requisitos": Álgebra Linear, Cálculo e Python.
- Intruduzir duas das bibliotecas livres em cada uma das áreas :
  - Fenics (`https://fenicsproject.org/`),
  - Tensorflow (`https://www.tensorflow.org/`).
- Repositório do Workshop ($\geq$ 10/02/2021 comentado) `https://github.com/felipefr/galerkinML_EAMC2021.git`.
- Apresentar um exemplo que une as duas áreas utilizando Bases Reduzidas.
- Sugerir literaturas mais avançadas.

Outline

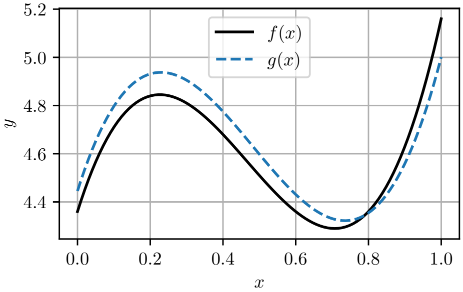
Problema de Aproximação de Funções

- Quem (= Espaço de dimensão inifinita)? Ex: Seja $f : [0,1] \to \mathbb{R}$, $f \in U = C^1(0,1)$ (Funções contínuas, integráveis, outras derivadas contínuas, etc).



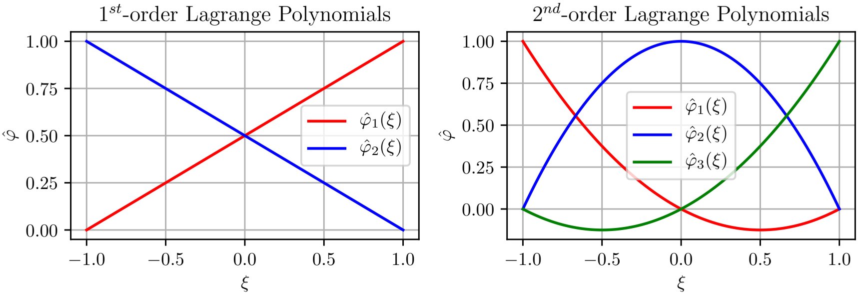
Problema de Aproximação de Funções

- Quem (= Espaço de dimensão inifinita)? Ex: Seja $f : [0,1] \to \mathbb{R}$, $f \in U = C^1(0,1)$ (Funções contínuas, integráveis, outras derivadas contínuas, etc).
- Qual o sentido (=Norma)? Ex: $(f,g) = \int_0^1 fg\mathrm{d}x$, i.e., $\|\cdot\| = \sqrt{(\cdot,\cdot)}$



Problema de Aproximação de Funções

- Quem (= Espaço de dimensão inifinita)? Ex: Seja $f : [0,1] \to \mathbb{R}$, $f \in U = C^1(0,1)$ (Funções contínuas, integráveis, outras derivadas contínuas, etc).
- Qual o sentido (=Norma)? Ex: $(f,g) = \int_0^1 fg \,\mathrm{d}x$, i.e., $\|\cdot\| = \sqrt{(\cdot,\cdot)}$
- Candidatas (= Espaço de dimensão finita)? $u_h \in U_h \subset U$, $U_h = \text{span}\{\phi_i\}_{i=1}^n$.



Problema de Aproximação de Funções

- Quem (= Espaço de dimensão inifinita)? Ex: Seja $f : [0,1] \to \mathbb{R}$, $f \in U = C^1(0,1)$ (Funções contínuas, integráveis, outras derivadas contínuas, etc).
- Qual o sentido (=Norma)? Ex: $(f,g) = \int_0^1 fg \mathrm{d}x$, i.e., $\|\cdot\| = \sqrt{(\cdot,\cdot)}$
- Candidatas (= Espaço de dimensão finita)? $u_h \in U_h \subset U$, $U_h = \mathrm{span}\{\phi_i\}_{i=1}^n$.
- Problema : Achar $u_h \in U_h$ tal que $\|u_h - f\| \leq \|v_h - f\| \, \forall v_h \in U_h$
- Resolução : Usando $u_h(x) = \sum_{j=1}^n \alpha_i \phi_i(x)$ temos:

$$(u_h - f, v_h) = 0 \Rightarrow (u_h, v_h) = (f, v_h) \, \forall v_h \in U_h$$

$$(u_h, \phi_i) = (f, \phi_i) \Rightarrow \sum_{j=1}^n (\phi_j, \phi_i)\alpha_j = (f, \phi_i) \, \forall i \in 1, \ldots, n \tag{1}$$

Problema de Poisson : Método de Galerkin

- Seja $\Omega = [0,1] \times [0,1] \subset \mathbb{R}^2$, achar $u : \Omega \to \mathbb{R}$ tal que

$$\begin{cases} -\Delta u & = f \quad \text{em}\, \Omega \\ u & = 0 \quad \text{em}\, \partial\Omega \end{cases} \tag{2}$$

- Seja $a(u,v) = \int_\Omega \nabla u \cdot \nabla v \mathrm{d}\Omega$ e $b(v) = \int_\Omega fv \mathrm{d}\Omega$, a condição equivalente a resolver (2) é achar $u \in U = \{w : \Omega \to \mathbb{R}\,; u|_{\partial\Omega} = 0 \text{ e "regular"}\}$ t.q.

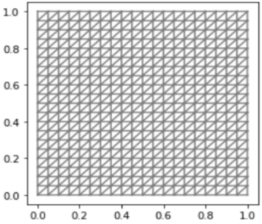
$$a(u,v) = b(v) \quad \forall v \in U. \tag{3}$$

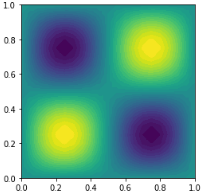
- Método de Galerkin: achar $u_h \in U_h$ t.q:

$$a(u_h, v_h) = b(v_h) \quad \forall v_h \in U_h \tag{4}$$

Problema de Poisson : Código base Fenics

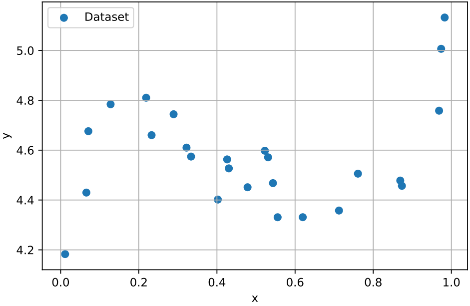
```
from dolfin import *
import matplotlib.pyplot as plt
import numpy as np

# Solves Poisson in a domain [0,1]x[0,1]
def solvePoisson(Nx):

    # Create mesh and define function space
    mesh = UnitSquareMesh(Nx,Nx)
    Uh = FunctionSpace(mesh, "Lagrange", 1)

    # Define Dirichlet boundary (the whole border)
    eps = DOLFIN_EPS
    def boundary(x):
        return np.min(x[0:2]) < eps or np.max(x[0:2]) > 1.0 - eps

    # Define boundary condition
    u0 = Constant(0.0)
    bc = DirichletBC(Uh, u0, boundary)

    # Define variational problem
    uh = TrialFunction(Uh)
    vh = TestFunction(Uh)
    f = Expression("sin(A*x[0])*sin(A*x[1])/(A*A)", degree=2, A = 2*np.pi)
    a = inner(grad(uh), grad(vh))*dx
    b = f*vh*dx

    # Compute solution
    uh = Function(Uh)
    solve(a == b, uh, bc)

    return uh

uh = solvePoisson(10)
# Plot solution
plt.figure(1)
plot(uh)
```





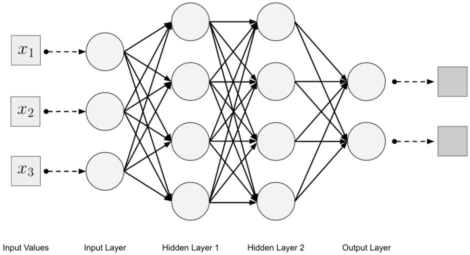
Redes Neurais (Regressão)

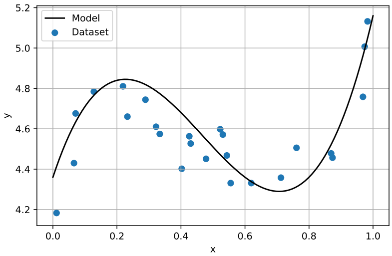
- Seja $D = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)});\ \mathbf{x}^{(i)} \in \mathbb{R}^n, \mathbf{y}^{(i)} \in \mathbb{R}^m, i = 1, \ldots, N_s\}$.



Redes Neurais (Regressão)

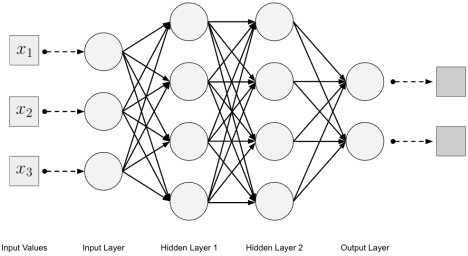
- Seja $D = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)});\ \mathbf{x}^{(i)} \in \mathbb{R}^n, \mathbf{y}^{(i)} \in \mathbb{R}^m, i = 1, \ldots, N_s\}$.
- Queremos achar $\mathcal{N}(\cdot, \boldsymbol{\theta}) : \mathbb{R}^n \to \mathbb{R}^m$, $\boldsymbol{\theta} \in \mathbb{R}^p$ tal que seja um "bom modelo" para descrever os dados.
- Problema: Achar $\boldsymbol{\theta} = \arg\min_{\boldsymbol{\omega}} \frac{1}{N_s} \sum_{i=1}^{N_s} \| \mathcal{N}(\mathbf{x}^{(i)}, \boldsymbol{\omega}) - \mathbf{y}^{(i)} \|_2^2$ (MSE).

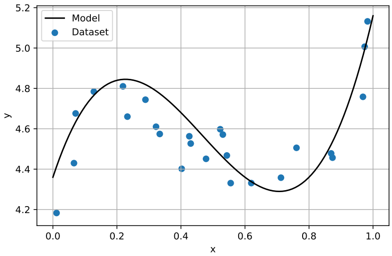




Redes Neurais (Regressão)

- Seja $D = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)});\ \mathbf{x}^{(i)} \in \mathbb{R}^n, \mathbf{y}^{(i)} \in \mathbb{R}^m, i = 1, \dots, N_s\}$.
- Queremos achar $\mathcal{N}(\cdot, \boldsymbol{\theta}) : \mathbb{R}^n \to \mathbb{R}^m$, $\boldsymbol{\theta} \in \mathbb{R}^p$ tal que seja um "bom modelo" para descrever os dados.
- Problema: Achar $\boldsymbol{\theta} = \arg\min_{\boldsymbol{\omega}} \dfrac{1}{N_s} \sum_{i=1}^{N_s} \| \mathcal{N}(\mathbf{x}^{(i)}, \boldsymbol{\omega}) - \mathbf{y}^{(i)} \|_2^2$ (MSE).
- Seja $m = n = 1$, o modelo mais simples é $\mathcal{N}(x, \boldsymbol{\theta}) = s(x, \boldsymbol{\theta}) = \theta_1 x + \theta_2$.





Redes Neurais (Regressão)

- Seja $D = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}); \mathbf{x}^{(i)} \in \mathbb{R}^n, \mathbf{y}^{(i)} \in \mathbb{R}^m, i = 1, \dots, N_s\}$.
- Queremos achar $\mathcal{N}(\cdot, \boldsymbol{\theta}) : \mathbb{R}^n \to \mathbb{R}^m$, $\boldsymbol{\theta} \in \mathbb{R}^p$ tal que seja um "bom modelo" para descrever os dados.
- Problema: Achar $\boldsymbol{\theta} = \arg\min_{\boldsymbol{\omega}} \dfrac{1}{N_s} \sum_{i=1}^{N_s} \| \mathcal{N}(\mathbf{x}^{(i)}, \boldsymbol{\omega}) - \mathbf{y}^{(i)} \|_2^2$ (MSE).
- Seja $m = n = 1$, o modelo mais simples é $\mathcal{N}(x, \boldsymbol{\theta}) = s(x, \boldsymbol{\theta}) = \theta_1 x + \theta_2$.
- Tome agora 2 neurônios na camada escondida

$$\mathcal{N}(x, \boldsymbol{\theta}) = s(x, \boldsymbol{\theta}^{(1)}) + s(x, \boldsymbol{\theta}^{(2)}) = (\theta_1^{(1)} + \theta_1^{(2)})x + (\theta_2^{(1)} + \theta_2^{(2)}) \quad (5)$$

Redes Neurais - Adcionando não-linearidades

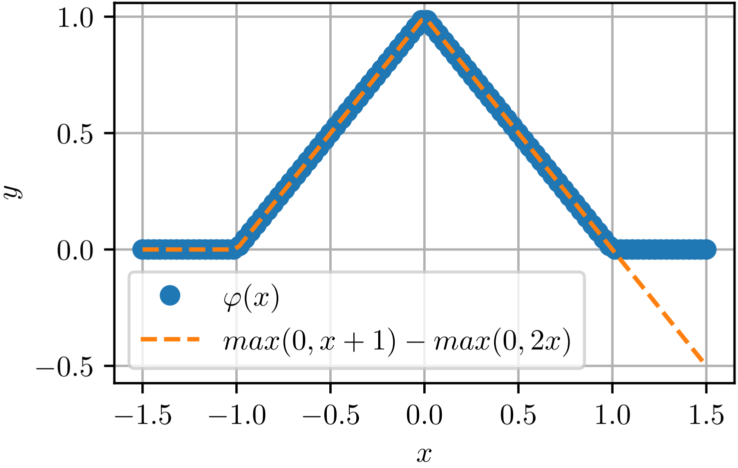
- Seja $\sigma : \mathbb{R} \to \mathbb{R}$, $\sigma(x) = \max(0, x)$ (ReLU).
- Redefinindo a rede

$$
\begin{aligned}
\mathcal{N}(\cdot, \boldsymbol{\theta}) &= \sigma \circ s(\cdot, \boldsymbol{\theta}^{(1)}) + \sigma \circ s(\cdot, \boldsymbol{\theta}^{(2)}) \\
\mathcal{N}(x, \boldsymbol{\theta}) &= \max(0, \theta_1^{(1)} x + \theta_2^{(1)}) + \max(0, \theta_1^{(2)} x + \theta_2^{(2)}) \\
&= \begin{cases}
\theta_1^{(1)} x + \theta_2^{(1)} & -\frac{\theta_2^{(1)}}{\theta_1^{(1)}} \leq x \leq -\frac{\theta_2^{(2)}}{\theta_1^{(2)}} \ \ (\textit{hip.}\ \frac{\theta_2^{(2)}}{\theta_1^{(2)}} \leq \frac{\theta_2^{(1)}}{\theta_1^{(1)}}) \\
(\theta_1^{(1)} + \theta_1^{(2)}) x + (\theta_2^{(1)} + \theta_2^{(2)}) & x \geq -\frac{\theta_2^{(2)}}{\theta_1^{(2)}} \\
0 & \text{caso contrário}
\end{cases}
\end{aligned}
$$

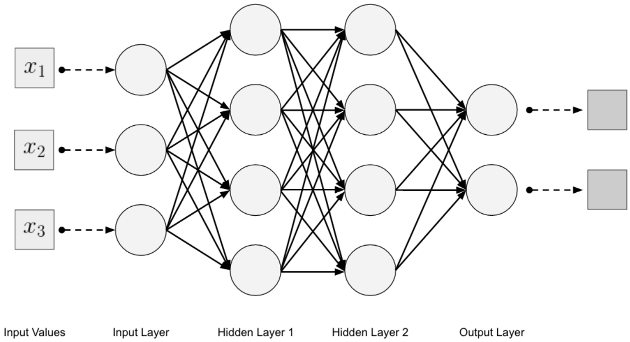
Redes Neurais - Espaço gerado

- Função "chapéu" (polinômio de Lagrange, base de $\mathbb{P}^1$).

$$\phi_0(x) = \max(0, x+1) + \max(0, 2x+0), \quad x \in [-1, 1] \tag{6}$$



Redes Neurais - Generalização



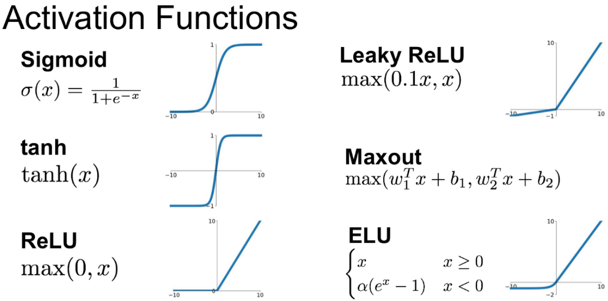
Redes Neurais - Generalização

- Seja L número de camadas internas
- n_ℓ, $\ell = 1, \ldots, L$, número de neurônios cada.
- $n_0 = n$, $n_{L+1} = m$ camadas de entrada e saída.
- $\begin{cases} \mathbf{s}_\ell(\cdot, \boldsymbol{\theta}^{(\ell)}) : & \mathbb{R}^{n_{\ell-1}} \to \mathbb{R}^{n_\ell} \\ & \mathbf{x} \mapsto \mathbf{s}_\ell(\mathbf{x}, \boldsymbol{\theta}^{(\ell)}) = \boldsymbol{\Theta}^{(\ell)}\mathbf{x} + \boldsymbol{\theta}_b^{(\ell)} \end{cases},$
  $\boldsymbol{\Theta}^{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}, \boldsymbol{\theta}_b^{(\ell)} \in \mathbb{R}^{n_\ell}$
- $\boldsymbol{\sigma} : \mathbb{R}^k \to \mathbb{R}^k$, $\boldsymbol{\sigma}(\mathbf{x}) = \begin{bmatrix} \sigma(x_1) & \sigma(x_2) & \ldots & \sigma(x_k) \end{bmatrix}^t$, k genérico.
- Para $\boldsymbol{\theta} = (\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(L)})$, $\mathcal{N}(\cdot, \boldsymbol{\theta}) : \mathbb{R}^n \to \mathbb{R}^m$.

$$\mathcal{N}(\cdot, \boldsymbol{\theta}) = \boldsymbol{\sigma} \circ \mathbf{s}_{L+1}(\cdot, \boldsymbol{\theta}^{(L+1)}) \circ \boldsymbol{\sigma} \circ \mathbf{s}_L(\cdot, \boldsymbol{\theta}^{(L)}) \circ \ldots \boldsymbol{\sigma} \circ \mathbf{s}_1(\cdot, \boldsymbol{\theta}^{(1)}) \quad (7)$$

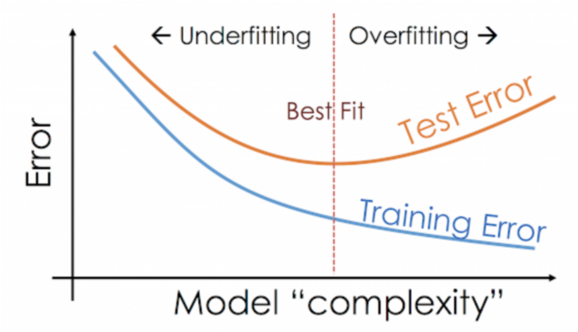
Redes Neurais - Considerações Práticas

- Diferentes funções de ativação.



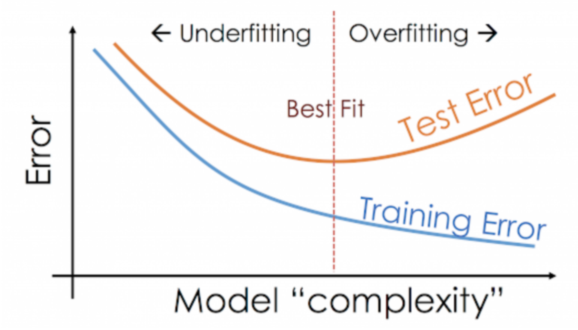
Redes Neurais - Considerações Práticas

- Diferentes funções de ativação.
- Validação (overfit).



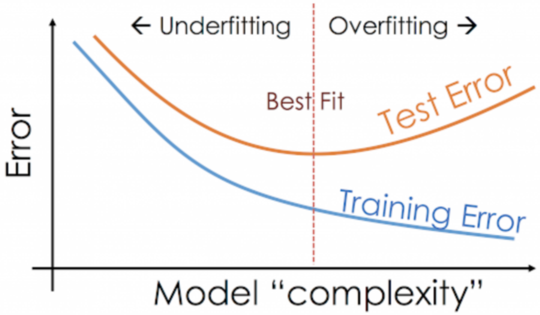
Redes Neurais - Considerações Práticas

- Diferentes funções de ativação.
- Validação (overfit).
- Normalização.



Redes Neurais - Considerações Práticas

- Diferentes funções de ativação.
- Validação (overfit).
- Normalização.
- Batch, Gradiente Estocástico, ADAM.

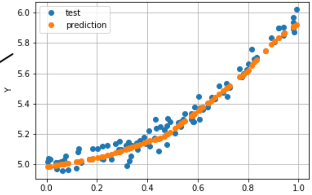


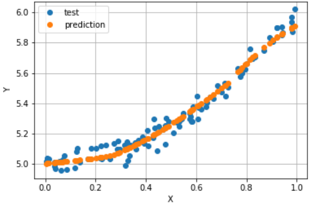
Redes Neurais - Código base Tensorflow

```
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import matplotlib.pyplot as plt

# Pretty print
class PrintDot(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs):
        if epoch % 100 == 0: print('')
        print('.', end='')

# Network settings
L = 2 # nb of hidden layers
n = 1 # input size
m = 1 # output size
Nneurons = 10 # nb neurons hidden layers
lr = 1.0e-3 # learning rate
EPOCHS = 100 # epochs to be trained
ratio_val = 0.2 # validation ration

labelfigs = '_L2_NN10_lr1m3'

# Creation of training dataset
Ns = 1000 # size dataset
deltaNoise = 0.05

np.random.seed(10)
X = np.random.rand(Ns)
Y = 5.0 + X*X + deltaNoise*np.random.randn(Ns)

X = X.reshape((Ns,n))
Y = Y.reshape((Ns,m))

scalerX = MinMaxScaler()
scalerX.fit(X)

scalerY = MinMaxScaler()
scalerY.fit(Y)

X_t = scalerX.transform(X)
Y_t = scalerY.transform(Y)

# Options activations: tf.nn.{tanh, sigmoid, leaky_relu, relu, linear}

# Building the architecture : L (hidden layers) + input + output layer
layers = [tf.keras.layers.Dense( Nneurons, activation=tf.keras.activations.linear, input_shape=(n,))]
for i in range(L):
    layers.append(tf.keras.layers.Dense( Nneurons, activation=tf.nn.relu))
layers.append(tf.keras.layers.Dense( m, activation=tf.nn.sigmoid ))

model = tf.keras.Sequential(layers)
```

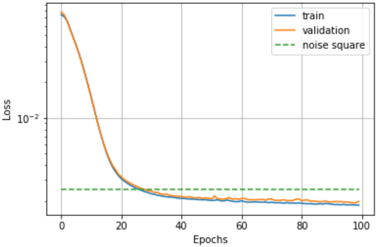
Redes Neurais - Código base Tensorflow

```
# Setting the opmisation algorithm
# Options optimisers: Adadelta, Adagrad, Adam, Adamax, FTRL, NAdam, RMSprop, SGD:
optimizer= tf.keras.optimizers.Adam(learning_rate = lr)
model.compile(loss = ['mse'], optimizer=optimizer, metrics=['mse','mae'])

# Fitting
hist = model.fit(X_t, Y_t, epochs=EPOCHS, validation_split=ratio_val, verbose=1, callbacks=[PrintDot() ], batch_size = 32)
model.save_weights('weights')
# model.load_weights('weights')


# Creation of test sample
Nt = 100
Xtest = np.random.rand(Nt)
Ytest = 5.0 + Xtest*Xtest + deltaNoise*np.random.randn(Nt)

Xtest = Xtest.reshape((Nt,n))
Ytest = Ytest.reshape((Nt,m))

Ypred = scalerY.inverse_transform( model.predict(scalerX.transform(Xtest)

# Plots
plt.figure(1)
plt.plot(hist.history['mse'], label = 'train')
plt.plot(hist.history['val_mse'], label = 'validation')
plt.plot([0,99],2*[deltaNoise**2], '--',  label = 'noise square' )
plt.grid()
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.yscale('log')
plt.legend()
plt.savefig('historic_{0}.png'.format(labelfigs))
plt.show()

plt.figure(2)
plt.plot(Xtest[:,0], Ytest[:,0], 'o', label = 'test')
plt.plot(Xtest[:,0], Ypred[:,0], 'o', label = 'prediction')
plt.grid()
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.savefig('prediction_{0}.png'.format(labelfigs))
plt.show()
```

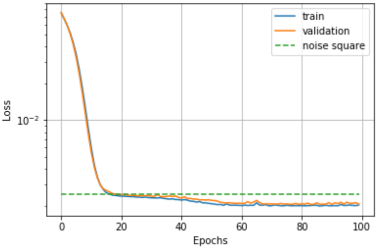




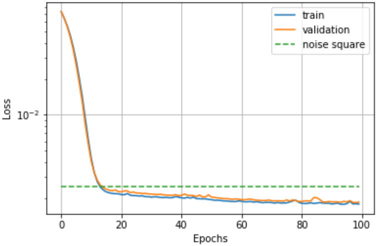
Treinamento para algumas arquiteturas



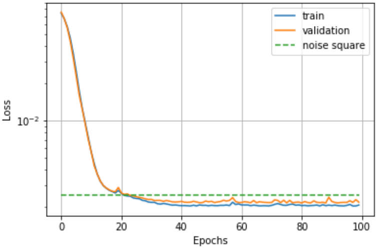
$L = 1, N_{neurons} = 5$



$L = 2, N_{neurons} = 5$



$L = 1, N_{neurons} = 10$



$L = 2, N_{neurons} = 10$

Bases Reduzidas

- Problema Discreto : Seja $\mathbf{A}(\boldsymbol{\mu})\mathbf{u}_h = \mathbf{f}_h(\boldsymbol{\mu})$ de dimensão N_h (grande).
- Motivação: Achar $\mathcal{N}(.,_{\boldsymbol{\theta}}) : \mathbb{R}^{N_\mu} \rightarrow \mathbb{R}^N$, com $N << N_h$, tal que $\mathbf{u}_h(\boldsymbol{\mu}) \approx \sum_{i=1}^{N} [\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\theta})]_i \boldsymbol{\xi}^{(i)}$

Bases Reduzidas

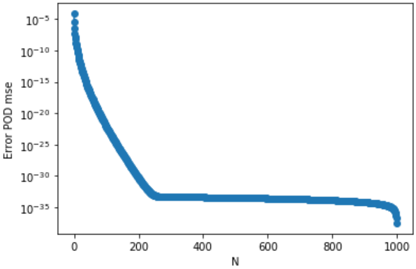
- Problema Discreto : Seja $\mathbf{A}(\boldsymbol{\mu})\mathbf{u}_h = \mathbf{f}_h(\boldsymbol{\mu})$ de dimensão N_h (grande).
- Motivação: Achar $\mathcal{N}(.,_{\boldsymbol{\theta}}) : \mathbb{R}^{N_\mu} \to \mathbb{R}^N$, com $N << N_h$, tal que $\mathbf{u}_h(\boldsymbol{\mu}) \approx \sum_{i=1}^{N} [\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\theta})]_i \boldsymbol{\xi}^{(i)}$
- Algoritmo:
  - Given $\mathbb{S} = \{\mathbf{u}^{(i)}\}_{i=1}^{N_s}$, with $N_s = 1000$, $\mathbf{u}^{(i)} \in L^2(\Omega)$
  - $\mathbf{C}$ is s.t. $C_{ij} = (\mathbf{u}^{(i)}, \mathbf{u}^{(j)})_{L^2(\Omega)}$, $1 \leq i, j \leq N_s^0$.
  - $\mathbf{C} = \mathbf{V} diag(\lambda_1, \ldots, \lambda_{N_s^0})\mathbf{V}^T$, $\mathbf{V}$ orthogonal.
  - for $i = 1, \ldots, N_{max}(= N_h)$ $\boldsymbol{\xi}^{(i)} = \frac{1}{\sqrt{\lambda_i}} \sum_{j=1}^{N_s^0} V_{ij}\mathbf{u}^{(j)}$, s.t $(\boldsymbol{\xi}^{(i)}, \boldsymbol{\xi}^{(j)})_{L^2(\Omega)} = \delta_{ij}$
  - $\Pi_N \mathbf{w} := \sum_{i=1}^{N} (\mathbf{w}, \boldsymbol{\xi}^{(i)})_{L^2(\Omega)}$
  - $\mathcal{E}_{POD}(N) = \sum_{j=N+1}^{N_{max}} \lambda_j = \sum_{i=1}^{N_s^0} \|\mathbf{u}^{(i)} - \Pi_N \mathbf{u}^{(i)}\|^2$
  - $\mathcal{E}_{POD}^{mse}(N) = \frac{1}{N_s} \sum_{j=N+1}^{N_{max}} \lambda_j$

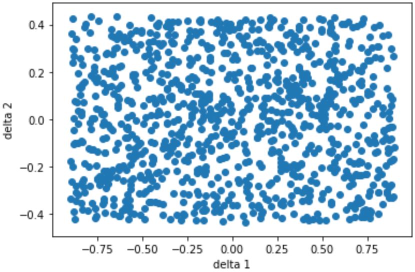
Poisson modificado com difusividade anisotrópica

- Seja $\Omega = [0,1] \times [0,1] \subset \mathbb{R}^2$, achar $u : \Omega \to \mathbb{R}$ tal que

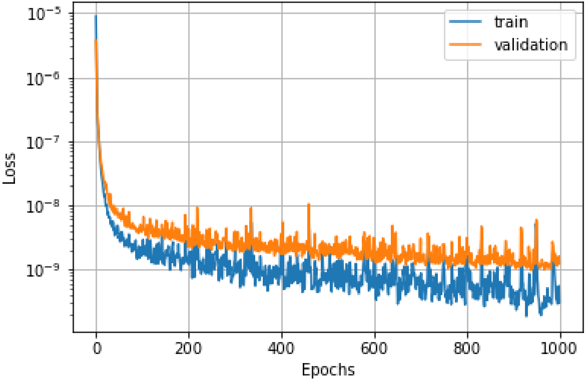
$$\begin{cases} -\operatorname{div}(K(\mu_1, \mu_2)\nabla u) & = f \quad \text{em } \Omega \\ u & = 0 \quad \text{em } \partial\Omega \end{cases} \tag{8}$$

com $K(\mu_1, \mu_2) = \begin{bmatrix} 1+\mu_1 & \mu_2 \\ \mu_2 & 1-\mu_1 \end{bmatrix}$, $\mu_1 \in [-0.9, 0.9]$, $\mu_2 \in [-0.43, 0.43]$, $N_s = 1000$.

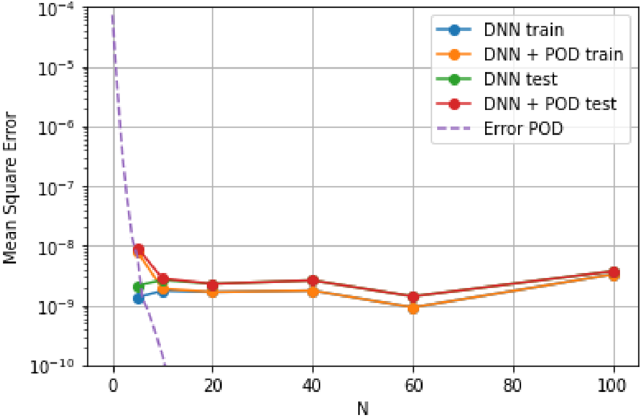




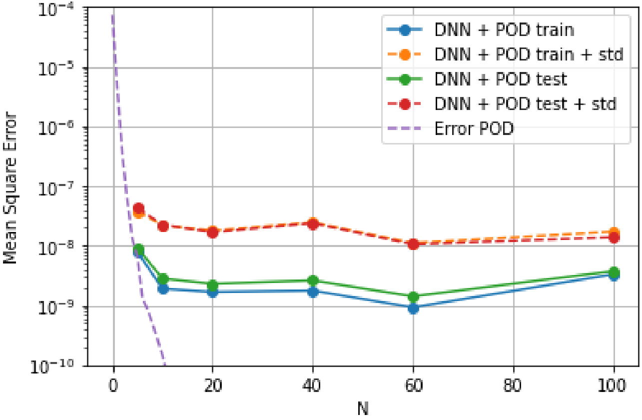
Treinamento para $N = 60$



Resultados $N \in [5, 10, 20, 40, 60, 80, 100]$



Resultados $N \in [5, 10, 20, 40, 60, 80, 100]$



Leitura Básica

1. Catherine F Higham and Desmond J Higham. “Deep learning: An introduction for applied mathematicians”. In:SIAM Review61.4(2019), pp. 860–891

   ⇒ *Revisão bastante clara e fácil de ler.*

2. Léon Bottou, Frank E. Curtis, and Jorge Nocedal. “Optimization Methods for Large-Scale Machine Learning”. In:SIAM Review60.2 (2018), pp. 223–311.doi:10.1137/16M1080173.3

   ⇒ *Paper de revisão com fundamentos de otimização não-linear com ênfase nos novos algoritmos que surgiram com o Deep Learning.*

3. Dal Santo, Deparis, Pegolotti, ”Data driven approximation of parametrized PDEs by reduced basis and neural networks”, 2020, JCP.

Artigos em teoria de aproximação para NN

1. A. Pinkus. "Approximation theory of the MLP model in neuralnetworks". In:Acta Numerica8 (1999), pp. 143–195.
   $\Rightarrow$ *Resultados para NN de uma camada. Prova densidade e resultados de aproximação usando C^∞. Também revisa a literatura dos anos 80 (aproximador universal).*
2. Weinan E, Chao Ma, and Lei Wu.Barron Spaces and the Compo-sitional Function Spaces for Neural Network Models. 2019. arXiv
   $\Rightarrow$ *(Re)define o conceito de espaços de Barron usando ReLU como ativação e múltiplas camadas.*
3. Ingo Gühring, Gitta Kutyniok, Philipp Petersen, "Error bounds for ap-proximations with deep ReLU neural networks in $W^{s,p}$ norm".In:Analysis and Applications18.05 (2020), pp. 803–859.doi:10.1142/S0219530519410021. arXiv: $\Rightarrow$ *Estuda a expressividade de NN ativações lineares por parte e estabelece limites inferiores e superiores para a complexidade da rede, i.e., número de camadas e neurônios, essencialmente no contexto de espaços de Sobolev em $W^{n,p}((0,1)^d)$.*

Obrigado!