

Rede Neural Informada pela Física avaliada para um problema de teste selecionado

CAP-425 Tópicos Avançados em Modelagem Ambiental

Eduardo Furlan Miranda

2023-11-30

Objetivos

- Implementação de um caso de teste (equação de viscosidade 1D de Burgers)
- Uso de abordagem PINN (aprendizado de máquina), rodando em GPU no supercomputador SDumont
 - A partir de um pequeno conjunto de observações dispersas, aproximar os parâmetros que melhor descrevem os dados
 - *“Inference problem” - “data-driven discovery of models”*
- Responder a pergunta:
 - Visando desempenho, qual é a melhor configuração de hiperparâmetros para este problema em particular?

Literatura (rede neural)

- Speedup de 7: código *Longwave Radiative Transfer* da ECMWF (*European Centre for Medium-Range Weather Forecasts*) parametrizando usando rede neural
 - Chevallier, F., Morcrette, J. -J., Chéruy, F., & Scott, N. A. (2000). **Use of a neural-network-based long-wave radiative-transfer scheme in the ECMWF atmospheric model**. *Quarterly Journal of the Royal Meteorological Society*, 126(563), 761–776. <https://doi.org/10.1002/qj.49712656318>
- Speedup de 10 a 10^5 : NN usada na parametrização de modelos físicos em modelos numéricos oceânicos e atmosféricos
 - Krasnopolsky, V. M., & Fox-Rabinovitz, M. S. (2006). **A new synergetic paradigm in environmental numerical modeling**: Hybrid models combining deterministic and machine learning components. *Ecological Modelling*, 191(1), 5–18. <https://doi.org/10.1016/j.ecolmodel.2005.08.009>

Histórico de implementações realizadas

- *INFERENCE* (a partir da equação obtenho os dados)
 - SDumont várias V100 em vários nós, MPI, comparando o *speedup* com relação ao método numérico
 - GColab, 1x GPU T4 comparando *speedup* com método numérico e variando hiperparâmetros
 - Sem resultados práticos, o método numérico vai ser sempre mais rápido do que o treinamento da rede
- *IDENTIFICATION* (a partir dos dados obtenho os parâmetros)
 - SDumont 1x V100 em um único nó variando hiperparâmetros e observando erros e tempos decorridos
 - Otimizar hiperparâmetros


Recursos e ambiente

- TensorFlow 1.15: biblioteca de código aberto para ML e IA
- JupyterLab: ambiente interativo e para a web
- SDumont: supercomputador do LNCC
 - Nó Sequana X: 2x Xeon 6152 22-core + 4x GPU Volta V100
 - Execução em apenas uma GPU

PINN

- Modelo de otimização que combina ANN e PDE
 - Uma das abordagens é **embutir a PDE na função de perda da ANN**
 - Converte o modelo em um problema de otimização
 - Pode gerar modelos substitutos aproximadores
 - Pode ser uma alternativa ou complemento ao método numérico
 - Pode usar precisão simples 32-bits nos modelos ← desempenho
- Aproveita-se a *diferenciação automática* do cálculo do gradiente
 - Usada no algoritmo de *backpropagation* para ajustar o modelo
 - Usa-se a *chain rule* para obter derivadas parciais de ordem arbitrária
- Pode resolver ou **aprender** da solução (*forward/inverse problems*)
- Dados de treinamento
 - *Pontos de colocação* para PDE, e *pontos interiores* para BC&IC
 - Pontos conhecidos obtidos por observação, métodos numéricos, etc.
 - Escolhidos para ajustar a PDE governante ao modelo
- Como ANN pode-se utilizar MLP, CNN, RNN, DeepONet, e outras
 - O problema de teste usa MLP

Minimizando o erro quadrático médio

- A PDE na MSE inclui os parâmetros da equação a serem obtidos
- Parâmetros da equação e da NN são obtidos minimizando o MSE
 (pesos, bias)

$$MSE = MSE_u + MSE_f$$

$$MSE_u = \frac{1}{N} \sum_{i=1}^N |u(t_u^i, x_u^i) - u^i|^2$$

Dados de treinamento para obtenção dos parâmetros da equação.
 Dados sintéticos gerados randomicamente por método numérico.

$$MSE_f = \frac{1}{N} \sum_{i=1}^N |f(t_u^i, x_u^i)|^2$$

Pontos de colocação para obtenção dos parâmetros da NN.
 Aplica a estrutura imposta pela equação.
 A quantidade e localização são as mesmas dos dados de treinamento.

Arquitetura MLP utilizada

- Função de ativação: tangente hiperbólica

- Os parâmetros da equação são obtidos durante o treinamento e otimização da função de perda.

- “Predição”

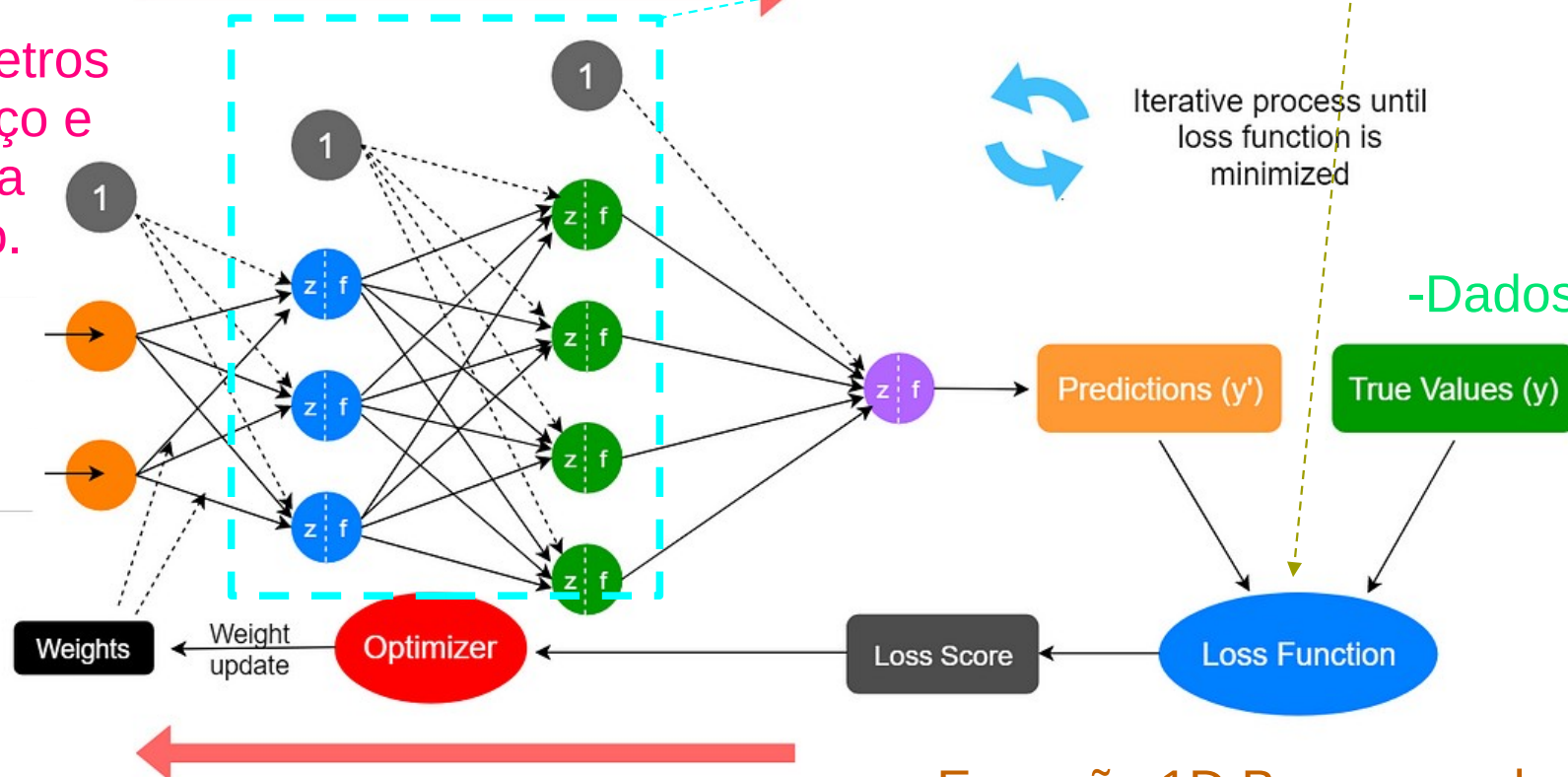
Forward Propagation

- Camadas escondidas.

- Parâmetros de espaço e tempo da equação.

x

t



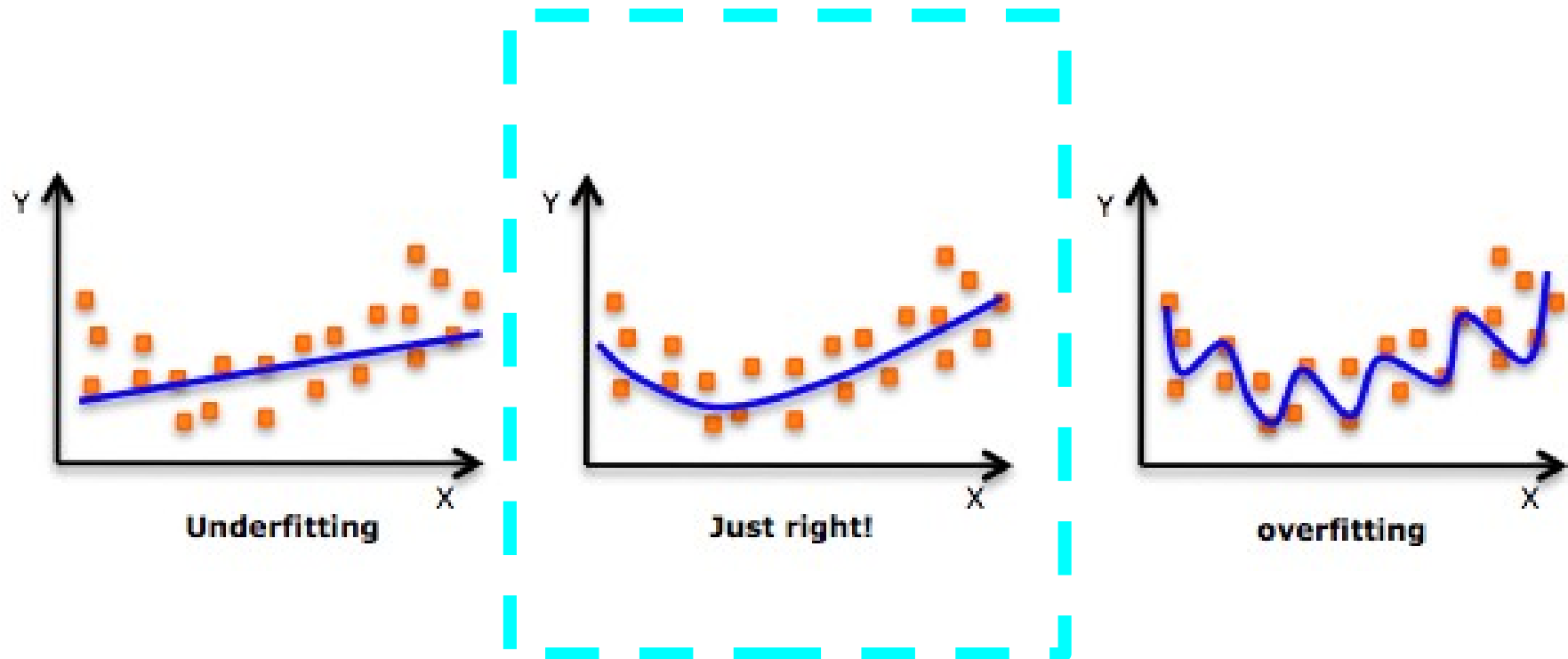
Backward Propagation

- Algoritmo de treinamento / otimização BFGS.

- Equação 1D Burgers embutida na função de perda.
 - Inclui os parâmetros a serem obtidos.
 - Neste caso não usa IC e BC.

Underfitting/Overfitting

- Overfitting: dificuldade de generalizar, e performance ruim
- Underfitting: dificuldade de modelar os dados, e performance ruim



Equação de Burgers

- PDE fundamental e uma equação de convecção-difusão
 - Pode ser derivada das equações de Navier-Stokes
- Usada em modelos de turbulência e teoria de ondas de choque, mecânica dos fluídos, acústica não linear, dinâmica dos gases, etc.

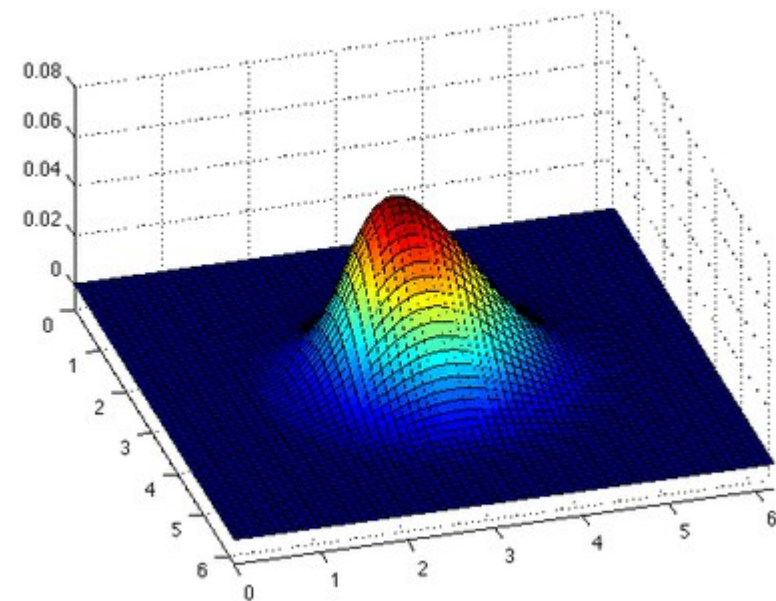
Velocidade do
fluído

Coeficiente de difusão
(ou viscosidade cinemática)

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}$$

Outra representação:

$$u_t + \lambda_1 u u_x - \lambda_2 u_{xx} = 0$$



Não é usado no problema inverso:

IC: $u(0, x) = -\sin(\pi x)$

BC: $u(t, -1) = u(t, 1) = 0$
for $-1.0 < x < +1.0$, and $0 < t$

Implementação (TensorFlow v1)

$$u = \lambda_1 = \text{"lambda_1"}$$

$$v = \lambda_2 = \text{"lambda_2"}$$

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}$$

método analítico

Os parâmetros da equação são embutidas na função de perda

```
def net_f(self, x, t):
    lambda_1 = self.lambda_1
    lambda_2 = tf.exp(self.lambda_2)
    u = self.net_u(x, t)
    u_t = tf.gradients(u, t)[0]
    u_x = tf.gradients(u, x)[0]
    u_xx = tf.gradients(u_x, x)[0]
    f = u_t + lambda_1 * u * u_x - lambda_2 * u_xx
    return f
```

API para
diferenciação
automática

Equação

Implementação

- Foram usados dados sintéticos, que podem gerados por um método numérico como por exemplo o método da quadratura de Gauss-Hermite
- Conjunto de dados de treinamento gerado aleatoriamente
 - $N_u = 2.000$ pontos 1D em todo o domínio espaço-temporal
 - Usado os parâmetros $\lambda_1 = 1,0$ e $\lambda_2 = 0,01/\pi$ para obter os dados sintéticos

PINN - Treinamento e Resultado

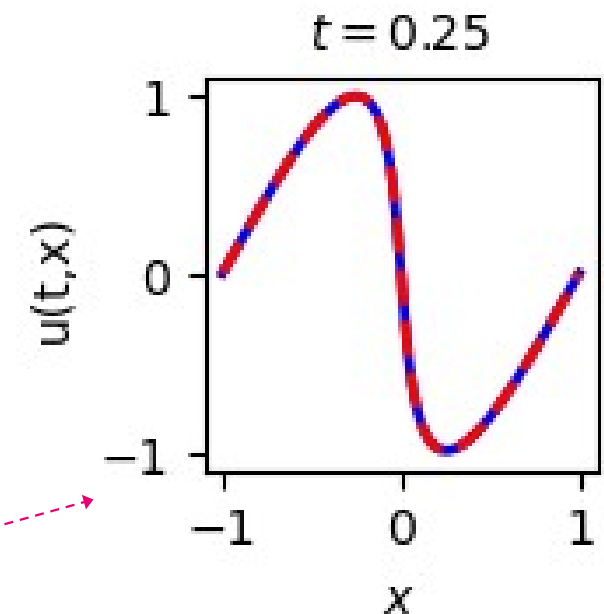
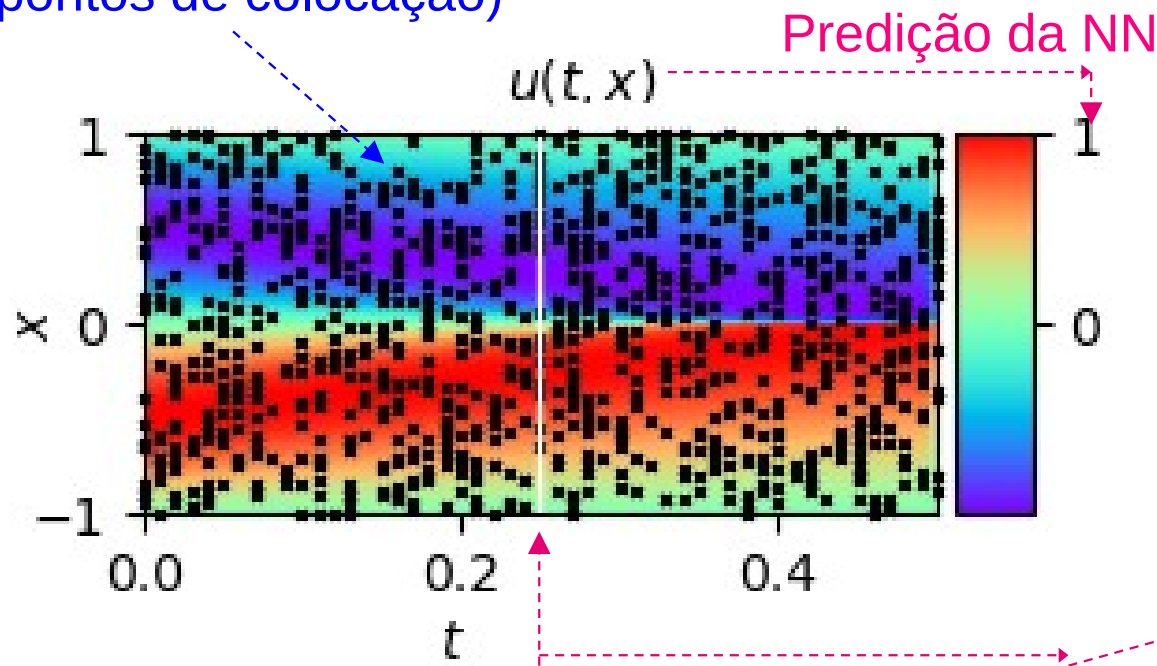
- Objetivo do treinamento: minimizar o Erro Médio Quadrático (MSE)

$$[u_t + 1uu_x - (0.01/x)u_{xx}]^2$$

Dados sintéticos de treinamento
(pontos de colocação)

$$[u_t + \lambda_1 uu_x - \lambda_2 u_{xx}]^2$$

PDE embutida na função
de perda da NN



Equação identificada:

$$u_t + 0.99916uu_x - 0.0032040u_{xx} = 0$$

Azul = método numérico
Vermelho = PINN

Algoritmo de otimização BFGS

- Método Broyden–Fletcher–Goldfarb–Shanno
 - Usado para encontrar zeros ou máximos e mínimos locais de funções, como alternativa ao método de Newton
 - Usa informações de curvatura para pré-condicionar o gradiente, e determinar a direção de descida
- No caso do problema de teste o desempenho é melhor que o método Adam (gradiente descendente estocástico)
 - Existem estudos que mostram outros métodos e técnicas que podem ser explorados
- Parâmetros: 50000 iterações ou λ próximo de 1, que é valor a ser obtido, usado nos dados sintéticos (a iteração pára quando a precisão dos números de ponto flutuante de hardware não é suficiente para continuar o treinamento).

Cálculo do Erro

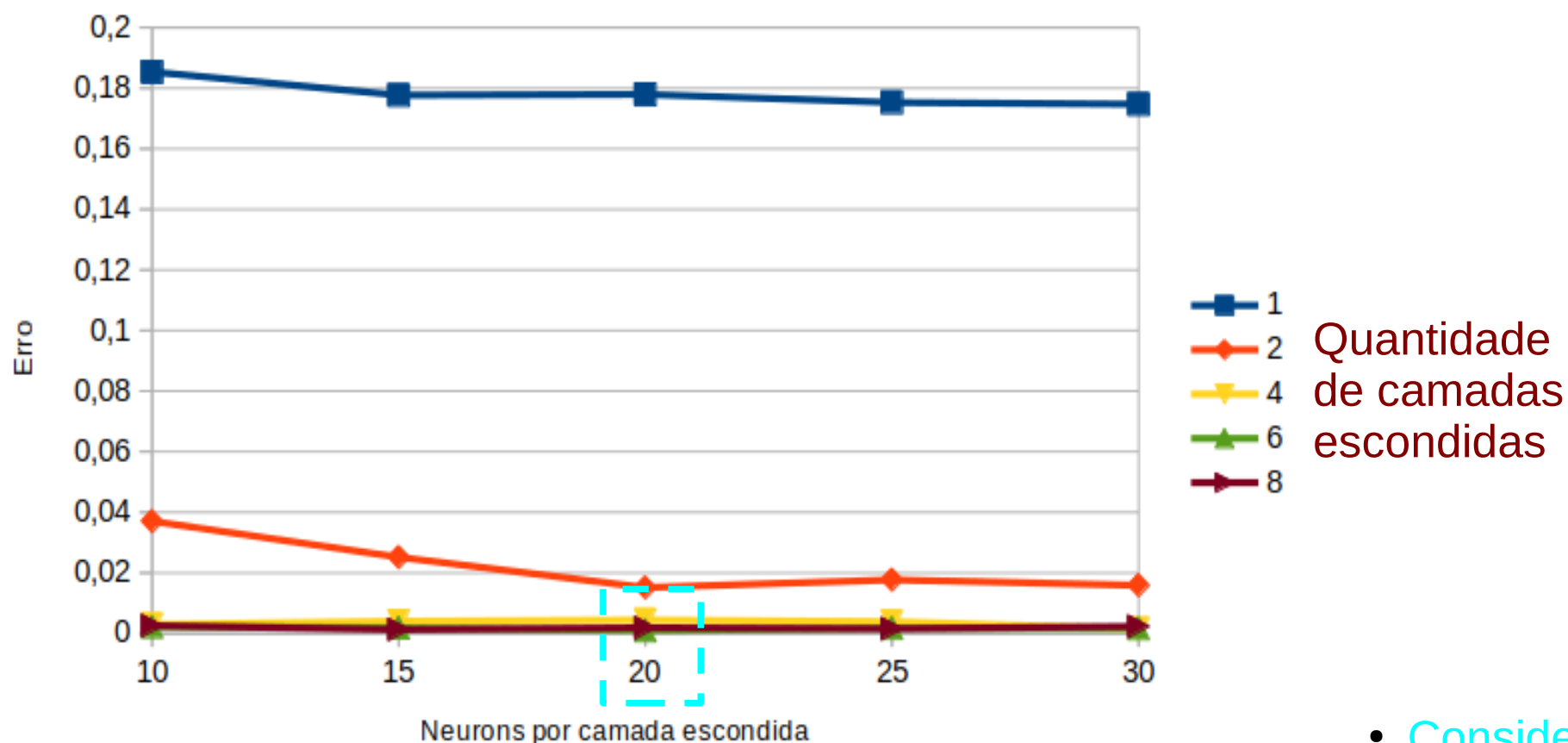
- Usa o método L2 norm (distância euclidiana entre dois conjuntos de pontos)

$$||x||_2 = \sqrt{\sum_{i=1}^n |x_i|^2}$$

- O termo “Erro” usado no trabalho é o erro relativo entre dados obtidos (u_pred) e sintéticos (u_star) usando a equação:

```
error_u=np.linalg.norm(u_star-u_pred,2)/np.linalg.norm(u_star,2)
```

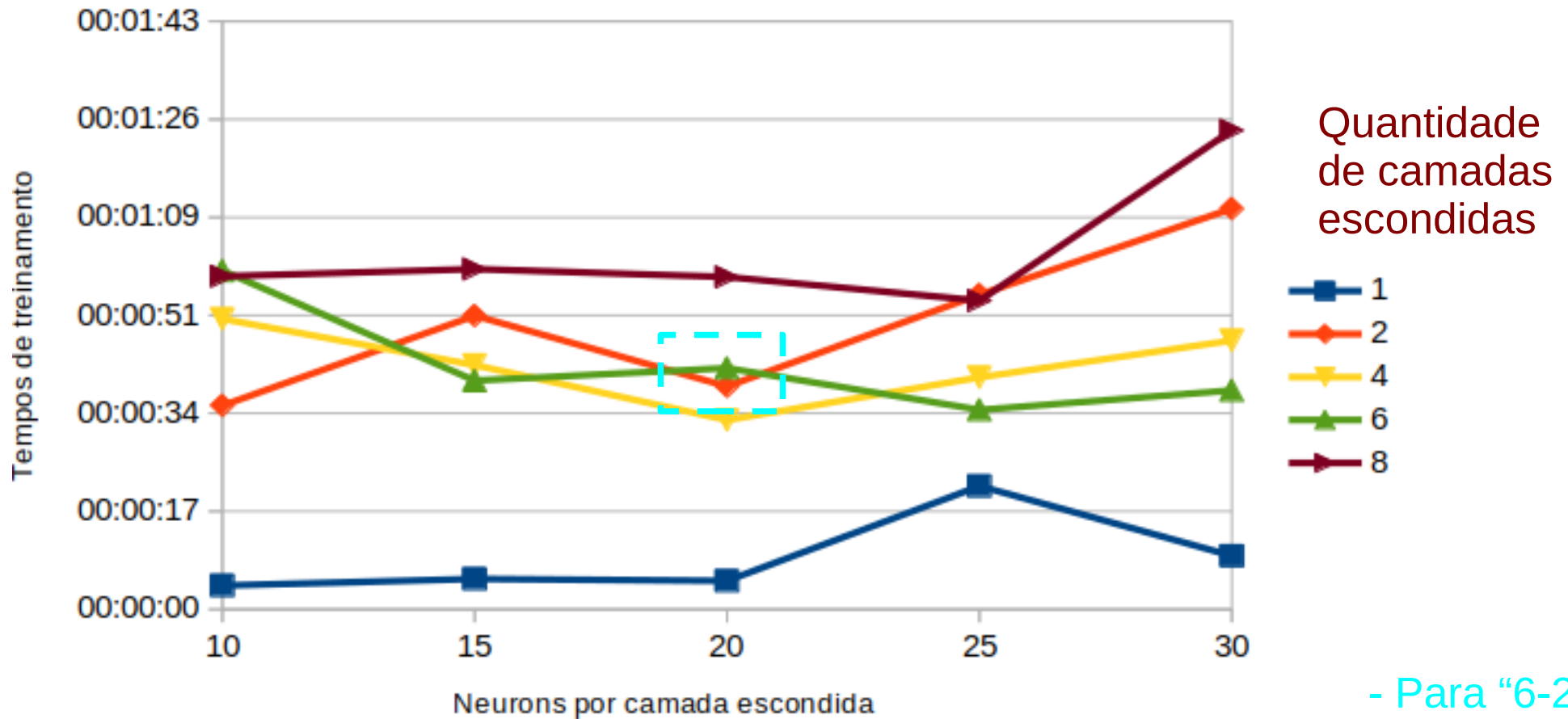
Erro L2 relativo, variando camadas e neurons



Nº Layers	Neurons por Layer				
	10	15	20	25	30
1	0,1853553	0,1776888	0,1779778	0,1752879	0,1747305
2	0,03703315	0,02519045	0,01515239	0,01771894	0,01589525
4	0,003002499	0,004058223	0,004434611	0,003920036	0,001564917
6	0,002214555	0,001881725	0,001005668	0,001818649	0,001706386
8	0,002578908	0,001305264	0,001897842	0,001572679	0,002326052

- Considerando apenas L2, a melhor opção seria 6 camadas e 20 neurons ("6-20")

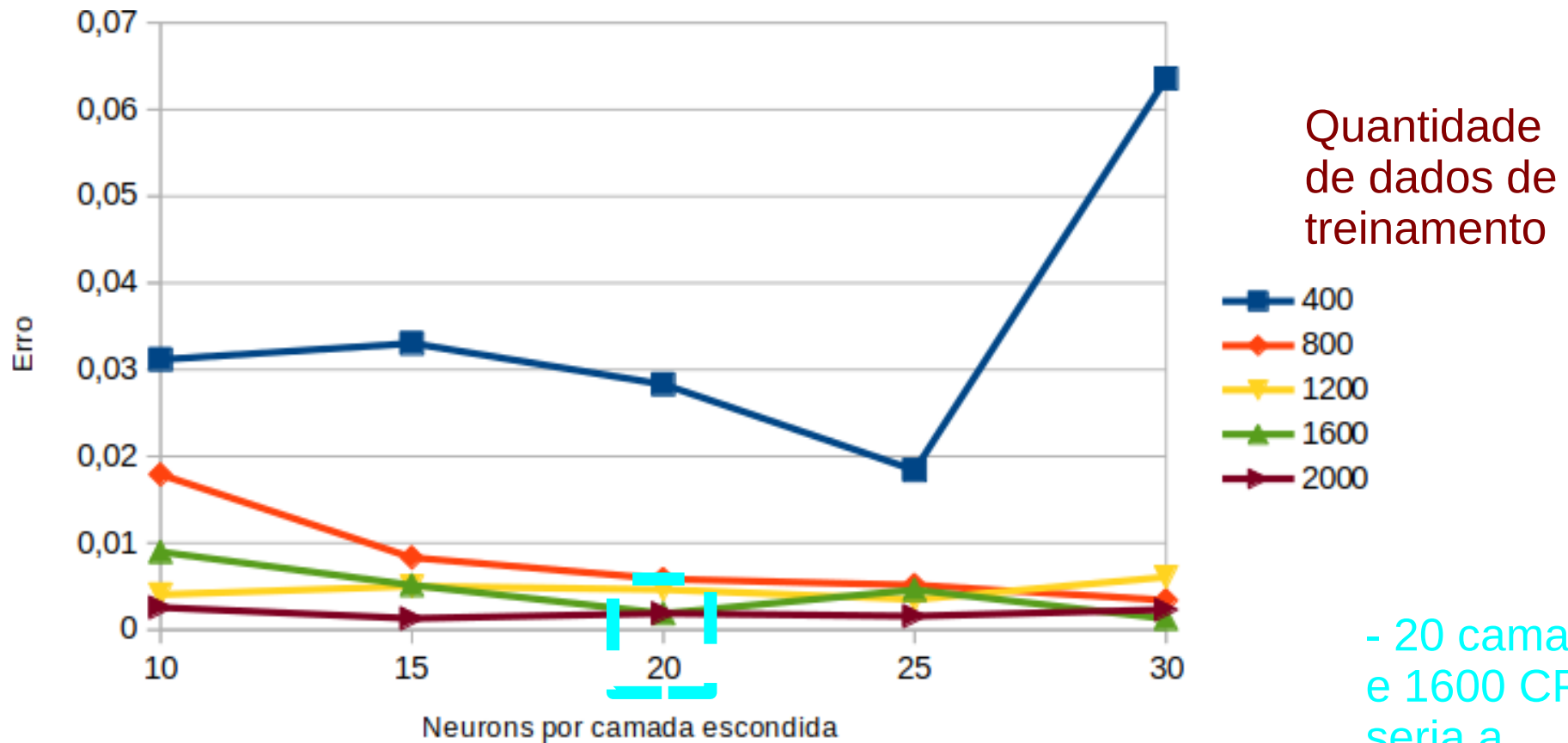
Tempos de treinamento (continuação do slide anterior)



- Para “6-20”
o tempos de
treinamento
para 2, 4, e
6 camadas
estão
próximos

Erro L2 relativo para 8 camadas escondidas 18

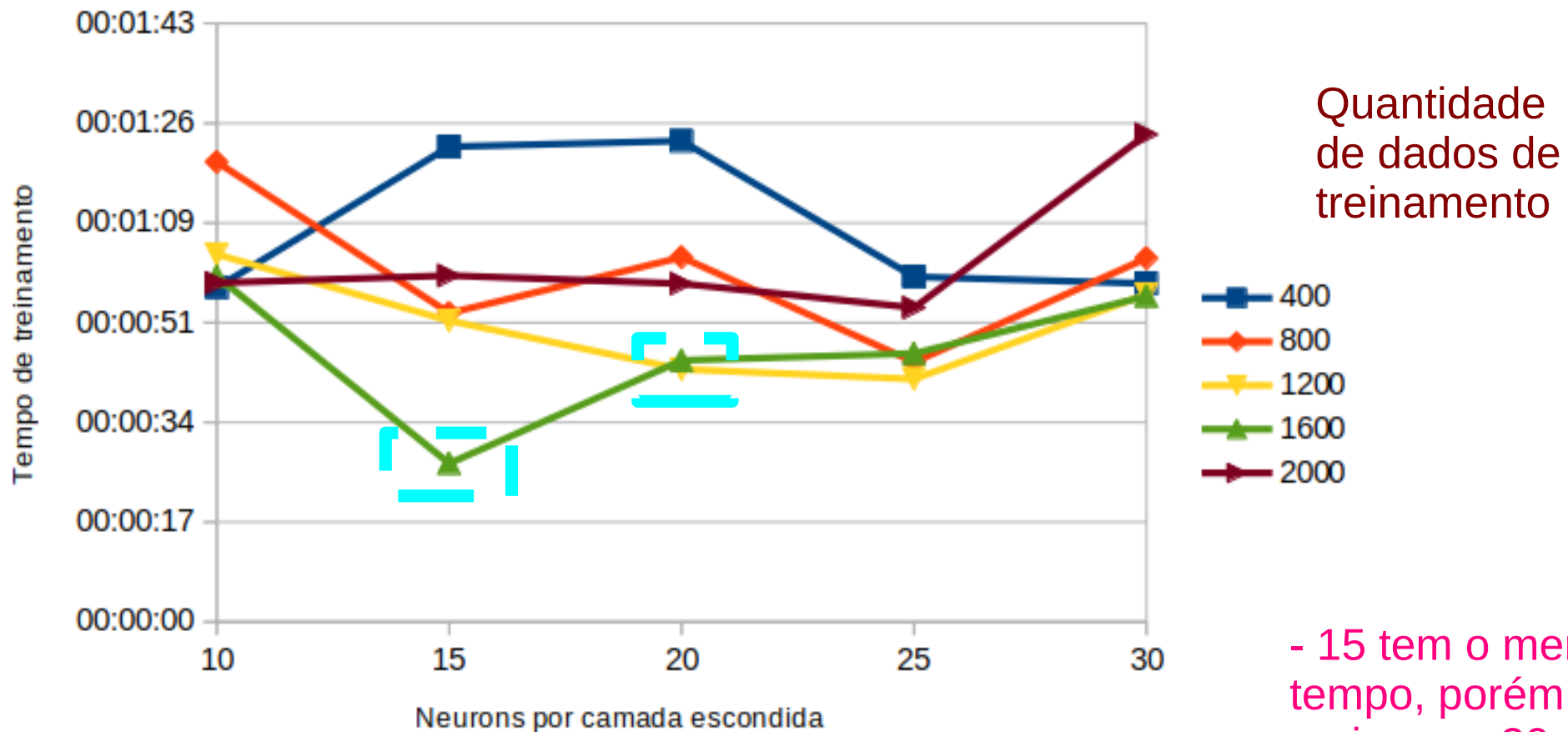
(fixando a quantidade de camadas escondidas, e variando a quantidade de dados de treinamento)



- 20 camadas e 1600 CP seria a melhor opção, pois para 30 o erro tende a aumentar

8 layers	Neurons por camada escondida				
Nu	10	15	20	25	30
400	0,03116806	0,03303246	0,02830449	0,01840675	0,06356454
800	0,01789704	0,0083338	0,005852155	0,005166975	0,003408238
1200	0,004067395	0,005027979	0,004636203	0,003503089	0,00608871
1600	0,008979423	0,005148291	0,001895437	0,00461331267	0,001266783
2000	0,002578908	0,001305264	0,001897842	0,001572679	0,002326052

Tempos de treinamento (continuação do slide anterior)

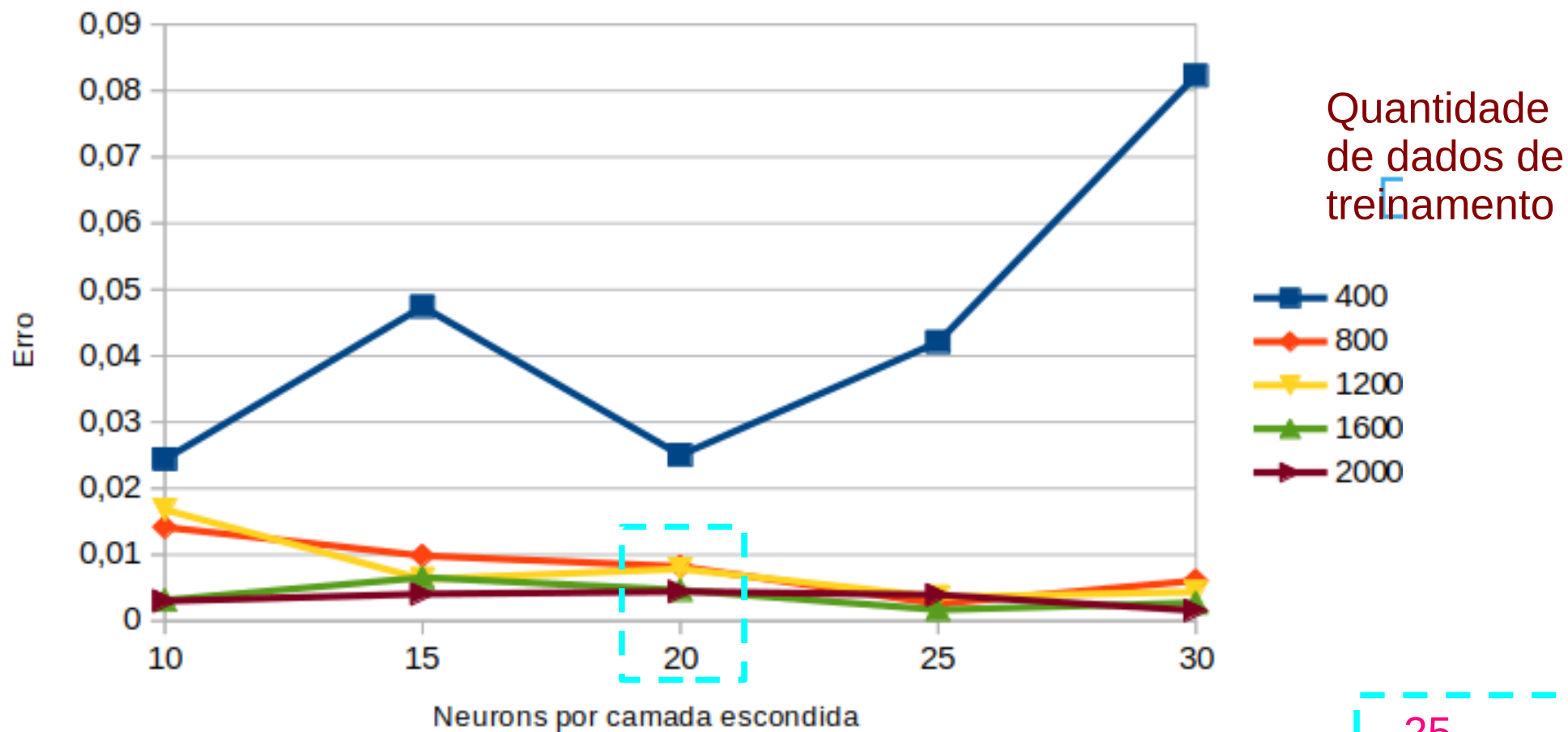


- 15 tem o menor tempo, porém L2 é maior que 20.
- 20 tem os tempos próximos dos demais.

8 layers	Neurons por camada escondida				
Nu	10	15	20	25	30
400	00:00:57	00:01:22	00:01:23	00:00:59	00:00:58
800	00:01:19	00:00:53	00:01:03	00:00:45	00:01:03
1200	00:01:03	00:00:52	00:00:43	00:00:42	00:00:56
1600	00:00:59	00:00:27	00:00:45	00:00:46	00:00:56
2000	00:00:58	00:01:00	00:00:58	00:00:54	00:01:24

Erro L2 relativo para 4 camadas escondidas 20

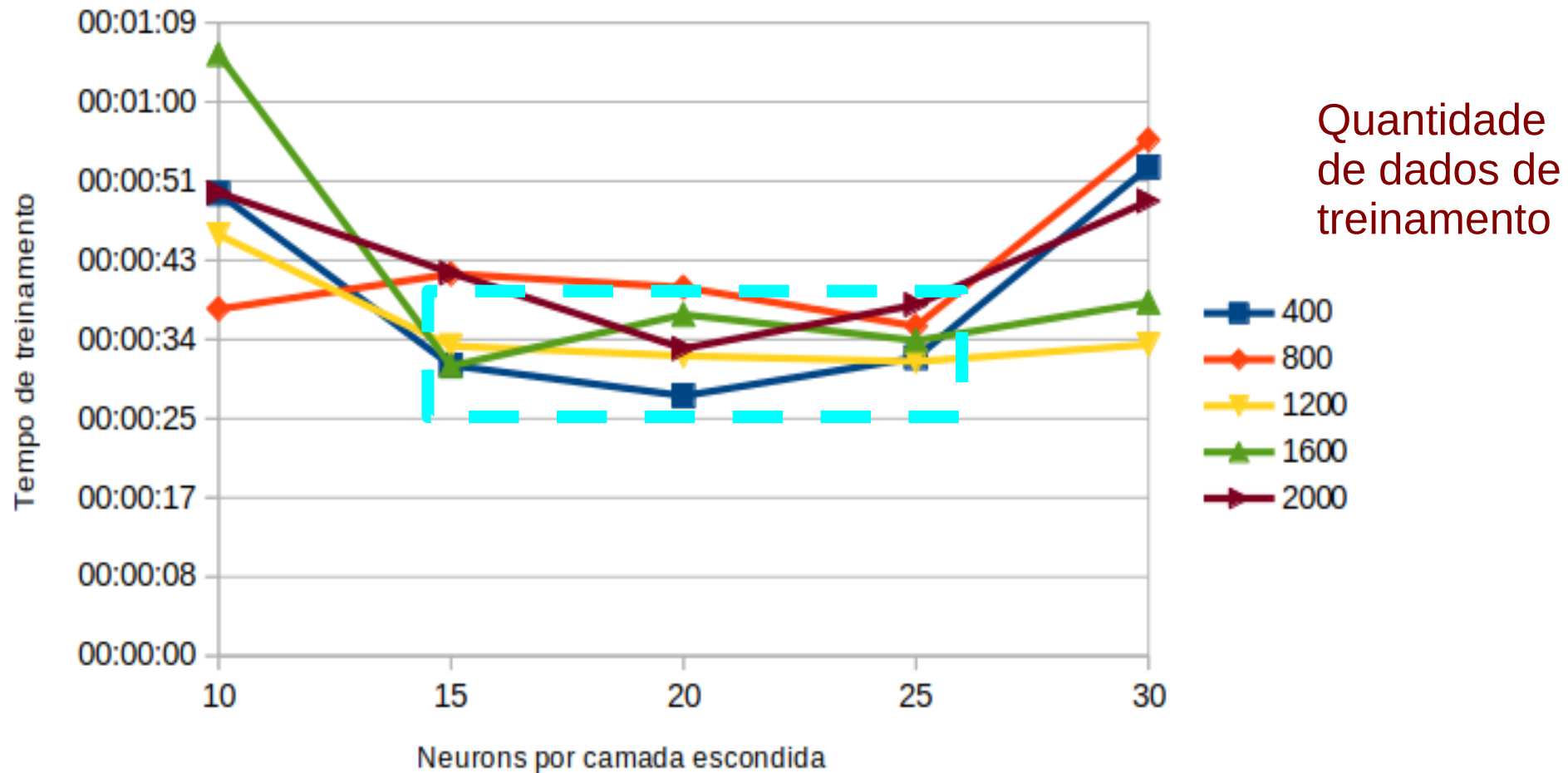
(fixando a quantidade de camadas escondidas, e variando a quantidade de dados de treinamento)



8 layers	Neurons por camada escondida				
Nu	10	15	20	25	30
400	0,02441041	0,04742153	0,02501846	0,04207384	0,08227192
800	0,01413198	0,009839402	0,00821884	0,002568378	0,006085704
1200	0,01677221	0,006306477	0,007846975	0,003610867	0,004376913
1600	0,003154124	0,006541028	0,004654613	0,001685301	0,002736355
2000	0,003002499	0,004058223	0,004434611	0,003920036	0,001564917

- 25
apresentou o
menor erro,
porém está
próximo de
20

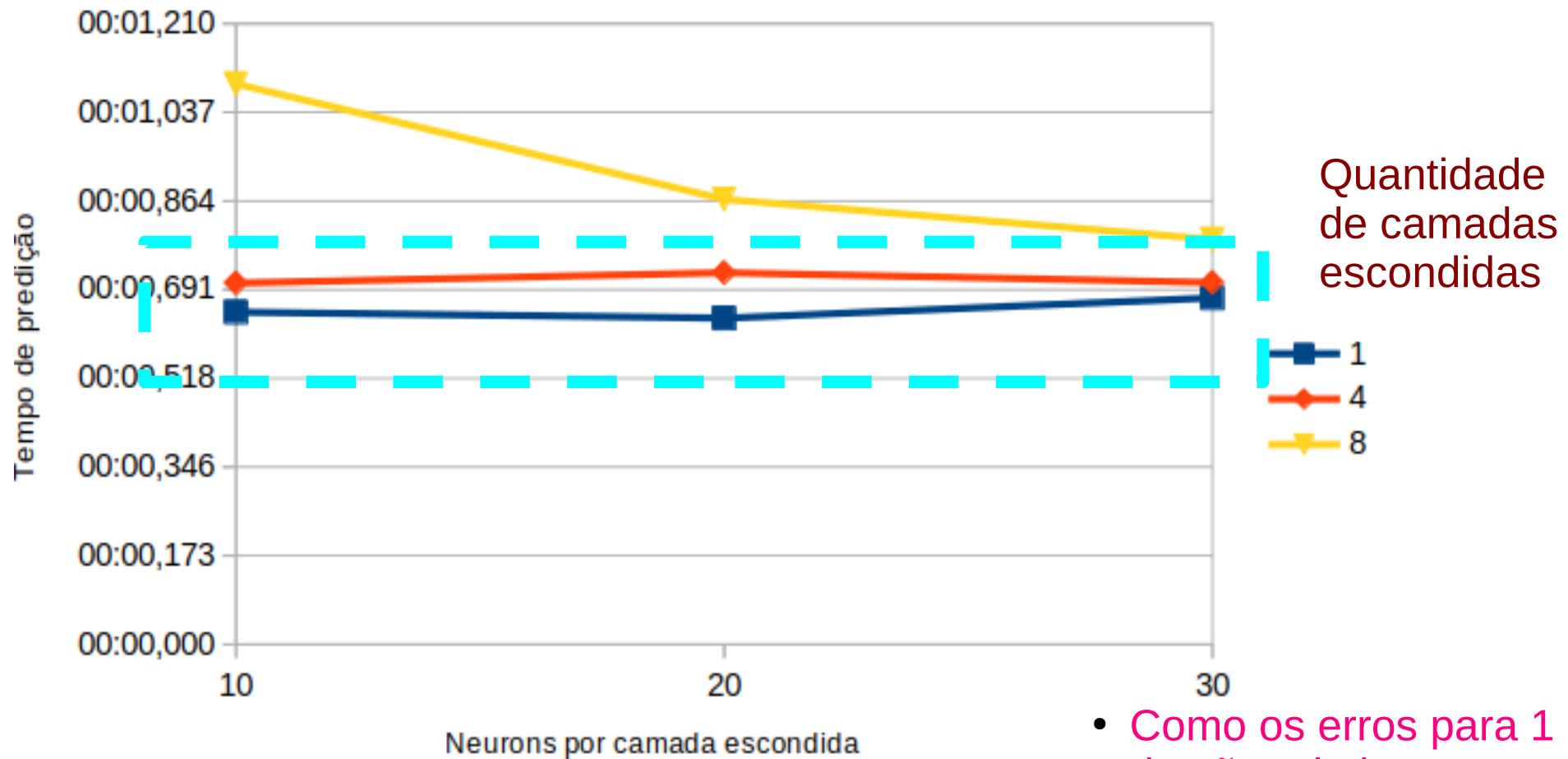
Tempos de treinamento (continuação do slide anterior)



8 layers	Neurons por camada escondida				
Nu	10	15	20	25	30
400	00:00:50	00:00:31	00:00:28	00:00:32	00:00:53
800	00:00:37	00:00:41	00:00:40	00:00:36	00:00:56
1200	00:00:45	00:00:33	00:00:32	00:00:32	00:00:34
1600	00:01:05	00:00:31	00:00:37	00:00:34	00:00:38
2000	00:00:50	00:00:41	00:00:33	00:00:38	00:00:49

Tempo de predição

Para alguns hiperparâmetros escolhidos



Nu=2000	Neurons por camada escondida		
Nº Layers	10	20	30
1	00:00,647	00:00,636	00:00,675
4	00:00,704	00:00,724	00:00,705
8	00:01,092	00:00,867	00:00,789

- Como os erros para 1 camada são relativamente grandes, a preferência é 4 e 8.
- Para 4 e 8 os tempos de predição apresentam pouca variação com relação à quantidade de neurons por camada.

Considerações finais

- No trabalho foi usado PINN em um problema inverso, de teste, que resolve a equação 1D de Burgers e usa dados sintéticos gerados por método numérico, para obter os parâmetros da equação.
- O teste foi executado no SDumont usando uma GPU V100.
- Os testes mostraram que para este caso específico os melhores hiperparâmetros são: $Nu=1600$, 6 camadas ocultas, e 20 ou 25 neurons por camada.
 - Serve como ponto de partida para estudos futuros.
 - Neste caso os tempos de treinamento ficaram em torno de 40 s, e os de predição em torno de 700 ms.

Trabalhos futuros

- Comparar o desempenho de PINN e método numérico rodando em várias GPUs/CPUs e em vários nós (MPI)
- Comparar o desempenho de algumas bibliotecas disponíveis
 - TF v1/v2, PyTorch, DeepXDE, e outras
- Melhorar o desempenho da otimização
 - Inicia o treinamento com o método Adam e termina com o L-BFGS-B
 - Outros algoritmos de treinamento
- Comparar desempenhos utilizando NN (maior quantidade de dados) e PINN (menor quantidade de dados), com métodos numéricos, para casos selecionados
- Comparar com outras arquiteturas de rede
- Adaptar o *Longwave Radiative Transfer* da ECMWF para o modelo eCrad/Monam, usando NN e PINN, e comparar os resultados

Obrigado

<https://github.com/efurlanm/425>