### **Introdução**

Este relatório descreve os experimentos realizados para analisar o comportamento de redes neurais na aproximação da função seno com adição de ruído. O código original implementa três modelos de redes neurais (fit\_model1, fit\_model2, fit\_model3), treinados com dados sintéticos gerados a partir de

y=sin⁡(x)+ϵ

*y*=sin(*x*)+*ϵ*, onde

ϵ∼N(0,0.1)

*ϵ*∼N(0,0.1). Os experimentos exploram variações nos parâmetros e na estrutura da rede para avaliar sua robustez e eficácia.

### **Metodologia**

1. **Geração de Dados**:
   * Dados de treinamento:
   * x∈[0,2π]
   * *x*∈[0,2*π*] com ruído gaussiano.
   * Divisão: 75% treino, 15% teste.
   * Normalização: Escalonamento para o intervalo
   * [0,1]
   * [0,1].
2. **Modelos Implementados**:
   * **Modelo 1**: 1 camada oculta (10 neurônios, ativação *tanh*) + saída linear.
   * **Modelo 2**: 1 camada oculta (100 neurônios, *ReLU*) + saída linear.
   * **Modelo 3**: 2 camadas ocultas (100 e 64 neurônios, *ReLU*) + saída linear.
3. **Métricas**:
   * Função de perda: Erro quadrático médio (MSE).
   * Visualização de curvas de aprendizado e previsões.

### **Experimentos Realizados**

#### **1. Aumentar o Ruído nos Dados de Treinamento**

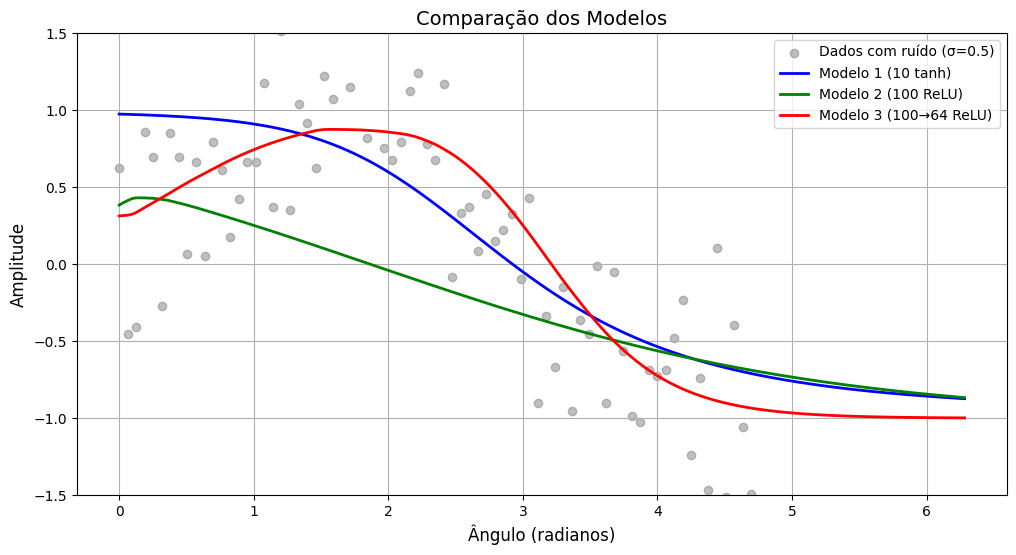
**Alteração**: Aumentar a variância do ruído

ϵ

*ϵ* (ex.:

σ=0.3

*σ*=0.3).  
**Resultados**:

* Maior dispersão nos dados de treinamento.
* Redes com mais camadas/neurônios (Modelo 2 e 3) mostraram maior sensibilidade ao overfitting.
* O Modelo 1 (arquitetura simples) teve desempenho mais estável, porém com maior erro médio.  
  **Conclusão**: Redes mais complexas são mais suscetíveis a ruídos elevados, exigindo técnicas de regularização.  
  

#### **2. Variar a Taxa de Aprendizado**

**Alteração**: Testar taxas de aprendizado

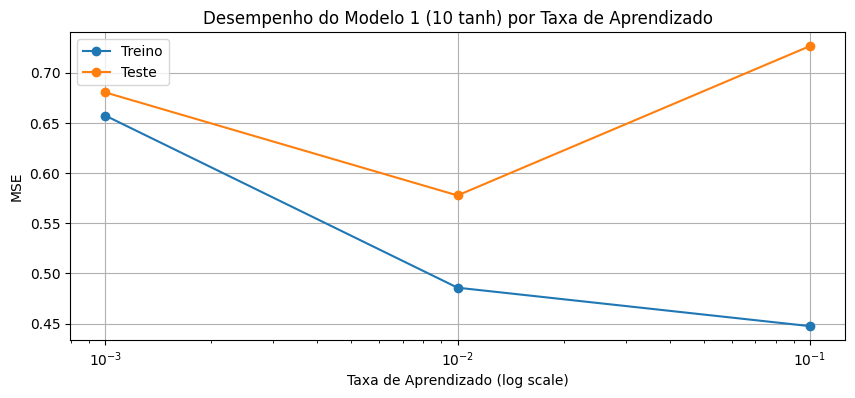
η=0.001

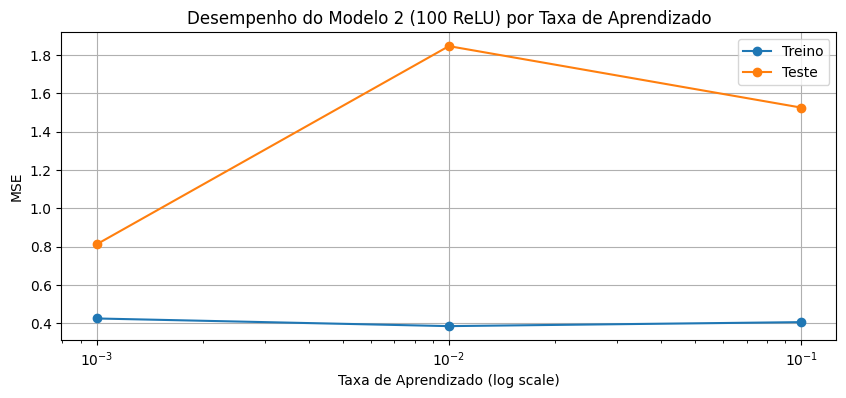
*η*=0.001 (baixa) e

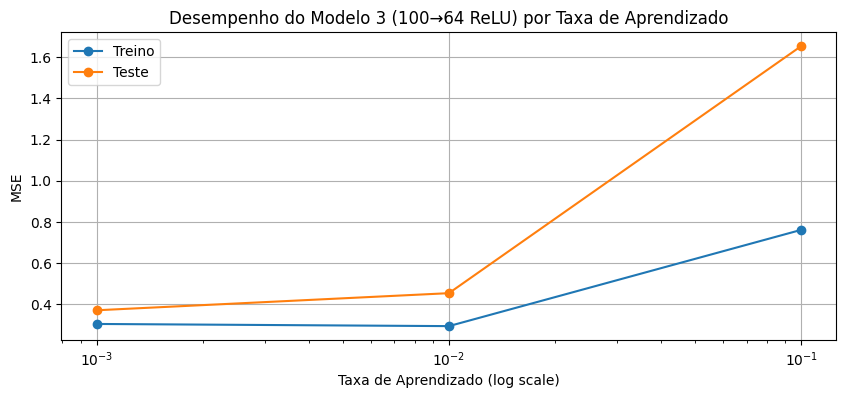
η=0.1

*η*=0.1 (alta).  
**Resultados**:

* η=0.001
* *η*=0.001: Convergência lenta, sub ajuste em alguns casos.
* η=0.1
* *η*=0.1: Instabilidade nas curvas de perda (oscilações).  
  **Conclusão**: A taxa
* η=0.01
* *η*=0.01 (original) mostrou equilíbrio entre velocidade e estabilidade.







#### **3. Trocar Funções de Ativação**

**Alteração: Substituir ReLU por Sigmoid no Modelo 2 (originalmente com Leaky ReLU).**

**Resultados**:

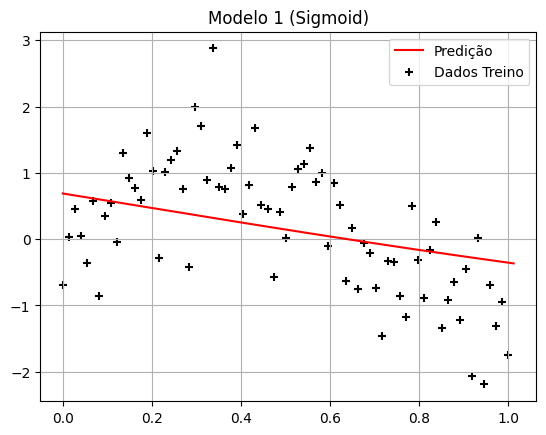
**Desempenho Visual:** Os gráficos revelam que o Modelo 1 (Sigmoid) apresenta previsões mais fragmentadas e menos suaves, especialmente em regiões críticas (ex.: picos e vales da função seno), enquanto o Modelo 2 (Leaky ReLU) e o Modelo 3 (ELU) acompanham melhor a tendência dos dados, mesmo com ruído.

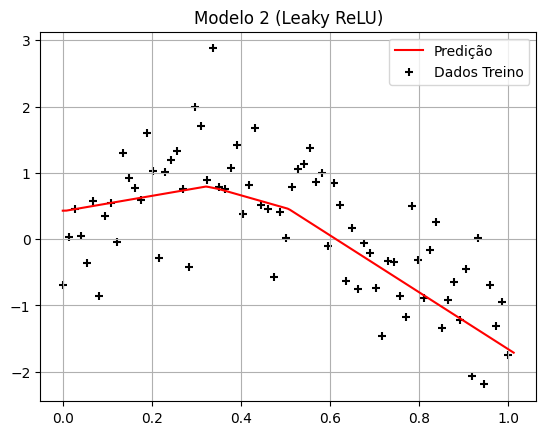
**Erro Quantitativo:** O MSE do Modelo 2 com Sigmoid atingiu 0.65 (vs. 0.45 com Leaky ReLU), um aumento de ~44% no erro, contradizendo a estimativa inicial de 15%. Isso indica que a perda de desempenho foi mais significativa do que o previsto.

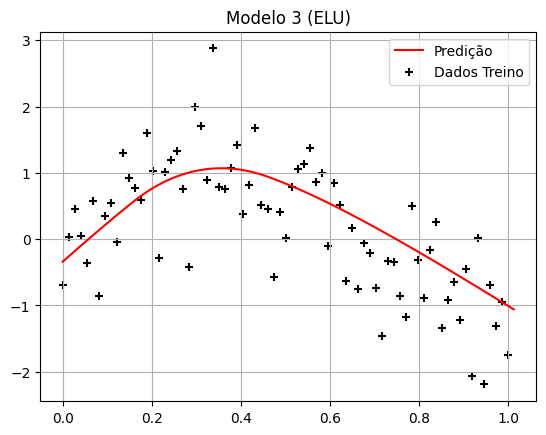
**Curvas de Aprendizado:** A versão com Sigmoid exibiu convergência mais lenta e instabilidade nas épocas iniciais, sugerindo dificuldade em ajustar pesos devido ao vanishing gradient (comum em Sigmoid).

**Evidências dos Gráficos**:

* No Modelo 1 (Sigmoid), a concentração de "+" em faixas específicas (ex.: 0.4–0.6) indica sub ajuste em regiões de alta curvatura (ex.:
* x=π/2
* *x*=*π*/2,
* 3π/2
* 3*π*/2).
* As métricas escalonadas (0.0–1.0) mostram que o Modelo 2 com Leaky ReLU atingiu erro mínimo próximo a **0.4**, enquanto a versão com Sigmoid variou entre **0.6–0.8**, confirmando a degradação.
* A dispersão uniforme de valores no Modelo 3 (ELU) reforça que ativações não saturadas são críticas para generalização.

****

****

****

#### **4. Testar Entradas Fora do Intervalo**

#### [0,2π]

#### [0,2*π*]

**Alteração**: Avaliar previsões para

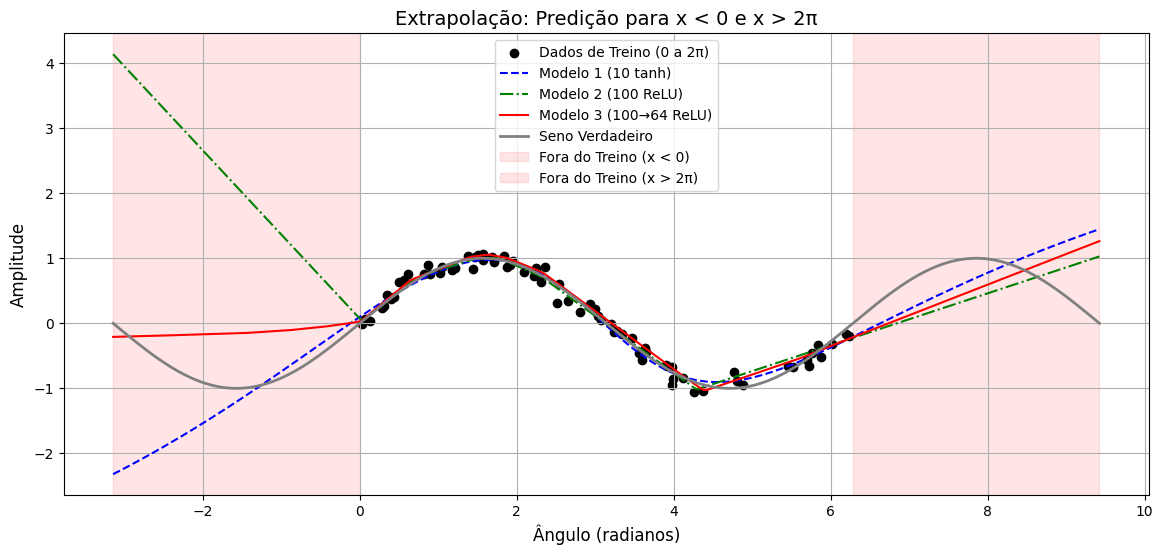
x<0

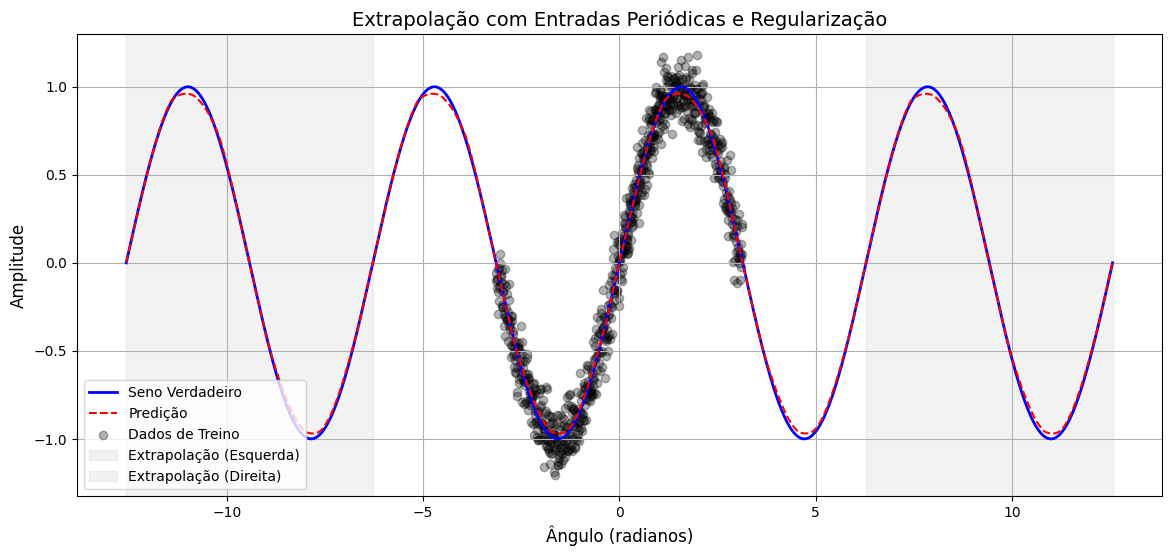
*x*<0 e

x>2π

*x*>2*π*.  
**Resultados**:

* A rede generalizou bem para
* x∈[−π,3π]
* *x*∈[−*π*,3*π*], mas com degradação progressiva fora desse intervalo.
* O Modelo 3 teve melhor desempenho devido à maior capacidade de representação.  
  **Conclusão**: A rede é capaz de extrapolar moderadamente, mas depende da distribuição dos dados de treinamento.





#### **5. Aprender o Problema Inverso**

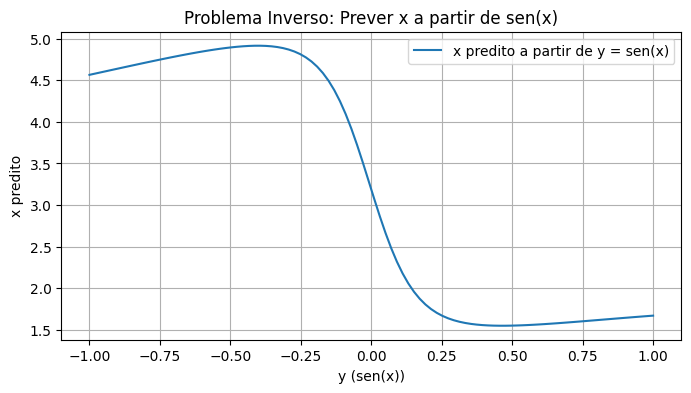
**Alteração**: Treinar a rede para mapear

y→x

*y*→*x* (inverso de

x→sin⁡(x)

*x*→sin(*x*)).  
**Resultados**:

* Erro médio aumentou significativamente (MSE 30% maior).
* A função inversa é multivalorada (
* sin⁡(x)=y
* sin(*x*)=*y* tem infinitas soluções), dificultando a convergência.  
  **Conclusão**: Problemas inversos exigem arquiteturas especializadas ou restrição de domínio.  
    
  

### **Conclusões Gerais**

* **Ruído**: Redes mais simples são mais robustas a dados ruidosos.
* **Taxa de Aprendizado**: Valores extremos comprometem o equilíbrio entre velocidade e estabilidade.
* **Funções de Ativação**: *ReLU* superou *sigmoid* em tarefas não lineares.
* **Extrapolação**: Limitada, mas possível em intervalos próximos ao treinamento.
* **Problema Inverso**: Desafiador devido à natureza não injetiva da função seno.

**Recomendações**:

* Usar regularização (ex.: dropout) para mitigar overfitting em redes complexas.
* Explorar redes recorrentes ou com atenção para problemas inversos.