# Universidade Federal de Santa Catarina
# Programa de Pós-Graduação em Ciência da Computação
# Inteligência Artificial Conexionista

Augusto André Souza Berndt 202000735

**Exercício 4.2**

Este exercício não segue mais a sequência de exercícios da coursera. O enunciado faz a seguinte sugestão de alterações no código:

1. Aumentar o "ruído" dos dados de treinamento e ver como a rede se comporta com níveis de variância nos dados de treinamento;
2. Aumentar e/ou diminuir a taxa de aprendizado;
3. Trocar as funções de ativação das camadas da rede neural
4. Ver como a rede se comporta para prever o valor para entradas menores que 0 (zero) e maiores que 2*pi;
5. Aprender o problema inverso (trocar a entrada pelas saídas e vice-versa);

Antes de iniciar a experimentação com o código fornecido mantemos aqui como referência todos os dados por padrão que vieram no código para análise posterior. Onde as variável de parâmetro das três funções principais de criação e treinamento das redes é dado por:

- ☐ **Epochs** o número máximo de épocas para aprendizado.
- ☐ **Learning rate** tanto ele quanto o próximo parâmetro são passados para a função SGD da biblioteca Keras. O learning rate definirá o “tamanho do passo” a cada epoch, LRs muito altos farão com que a atualização de pesos pelo gradiente seja mais abrupta, trazendo modificações maiores a cada epoch.
- ☐ **Momentum** é uma variável de entrada para realização do *stochastic gradient descent* com momentum com o Keras. Onde sua utilização acelera a convergência do gradiente na direção correta e reduz oscilações.
- ☐ **Patience** é o que causará uma parada no treinamento caso o treinamento atinja o número de épocas determinadas pela variável sem que haja melhoria no treinamento. Esta parada é feita com o método *EarlyStopping* do Keras.

Os valores por padrão que são dados para os parâmetros das criações e treinamento das 3 redes são apresentados a seguir:

- Modelo 1: epochs=200, learning rate=0.01, momentum=0.8, patience=100
- Modelo 2: epochs=600, learning rate=0.01, momentum=0.8, patience=100
- Modelo 3: epochs=1500, learning rate=0.01, momentum=0.8, patience=100

O modelo 1 é treinado com batches de 75, diferente das outras duas que são treinadas com batches de 5. Da configuração de neurônios dos 3 modelos:

- O modelo 1 é uma rede com uma única camada escondida completamente conectada composta de 10 neurônios com função de ativação tangente hiperbólica *tanh* e um neurônio na camada de saída com função de ativação linear.
- A segunda rede possui também uma única camada, mas composta de 100 neurônios com função de ativação relu e a camada de saída igual a anterior.
- O modelo 3 tem duas camadas escondidas, uma com 100 neurônios e outra com 64, ambas camadas com funções relu e um neurônio na camada de saída, sem função de ativação [*If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x).*]

Temos também as respectivas Figuras de 1 a 7 que apresentam as saídas padrões obtidas. Podemos notar claramente como o primeiro modelo não consegue aprender a função seno por ser limitada a uma função linear. Já os modelos 2 e 3 conseguem demonstrar um bom comportamento para aprender a função seno.

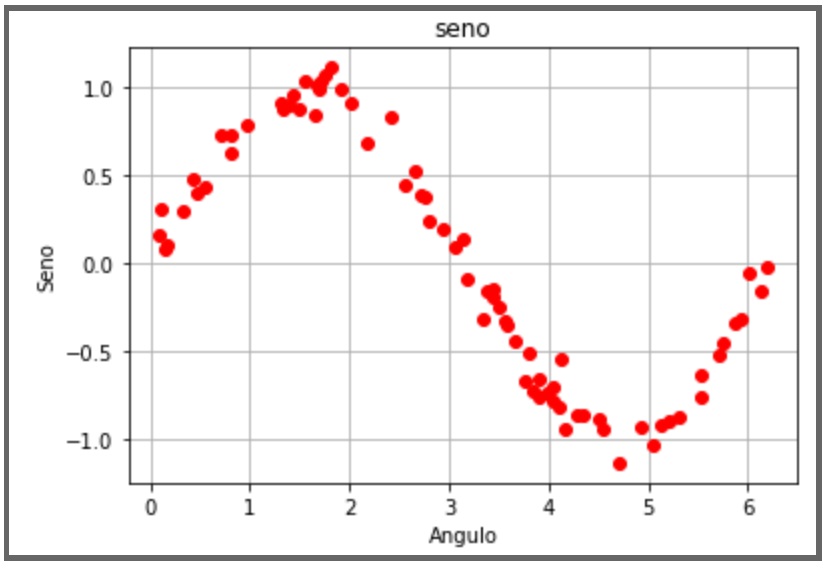


Figura 1 (dataset padrão).

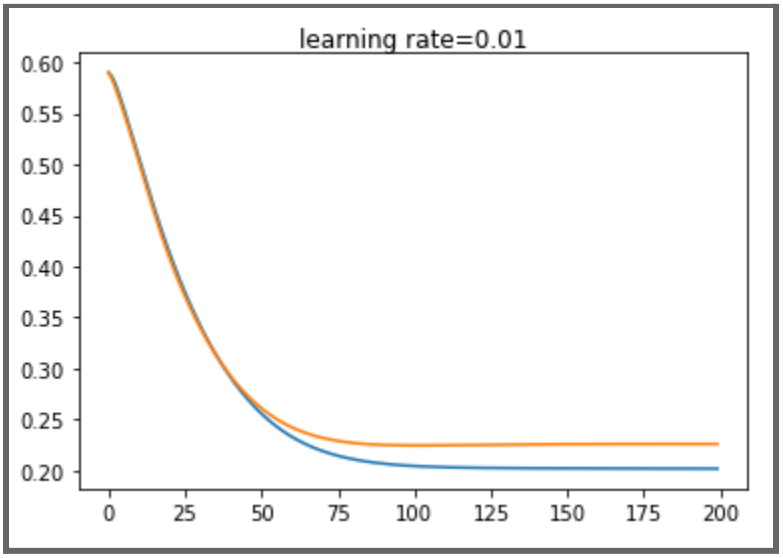


Figura 2 (aprendizado modelo 1 padrão).

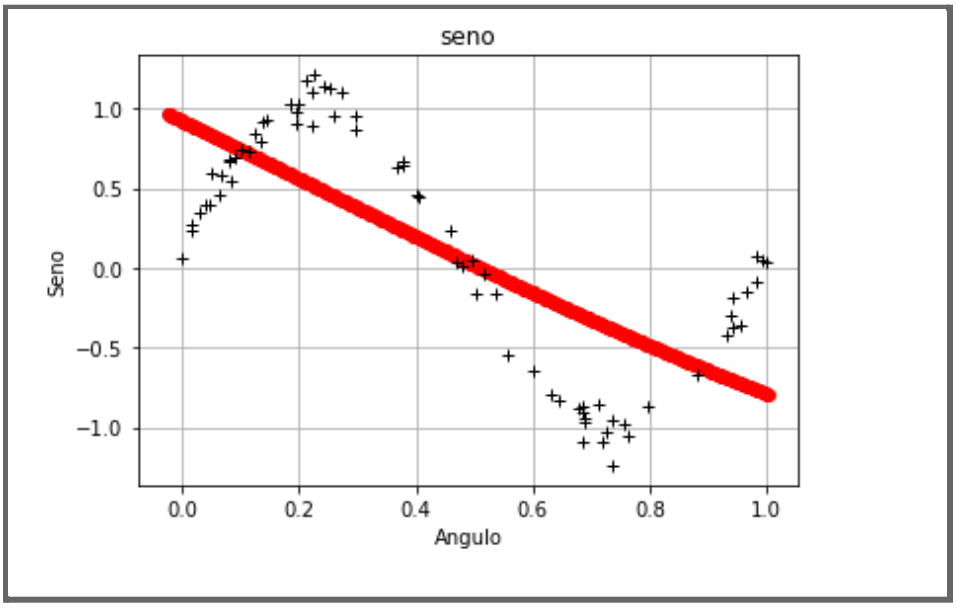


Figura 3 (classificação modelo 1 padrão).

```
model2 = fit_model2(xtrainN, ytrain, xtestN, ytest, 2000, 0.01, 0.8, 100)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 100)               200
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 101
=================================================================
Total params: 301
Trainable params: 301
Non-trainable params: 0
_________________________________________________________________
```

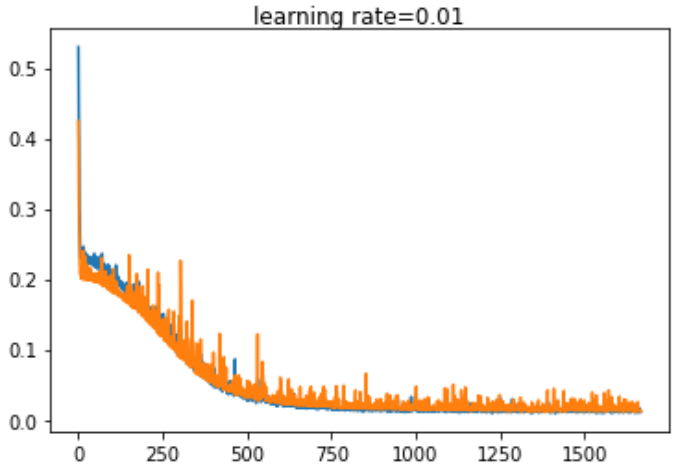


Figura 4 (aprendizado Modelo 2 padrão).

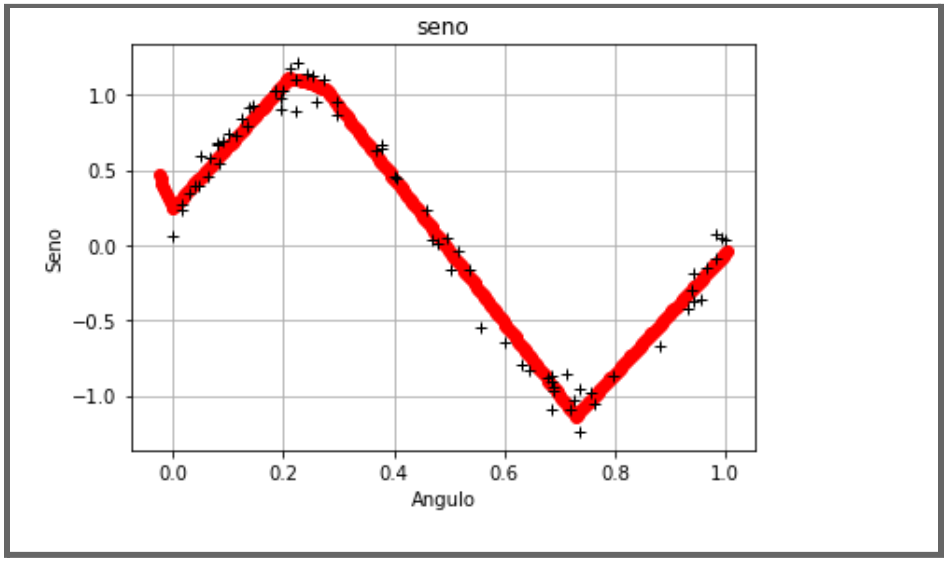


Figura 5 (classificação modelo 2 padrão).

```
model3 = fit_model3(xtrainN, ytrain, xtestN, ytest, 2000, 0.01,
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 100)               200
_________________________________________________________________
dense_5 (Dense)              (None, 64)                6464
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 65
=================================================================
Total params: 6,729
Trainable params: 6,729
Non-trainable params: 0
_________________________________________________________________
```

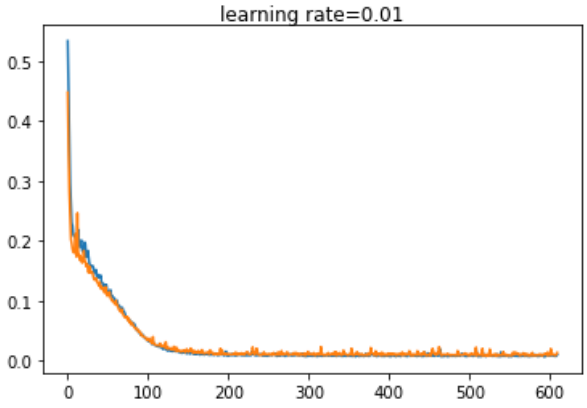


Figura 6 (aprendizado modelo 3 padrão).

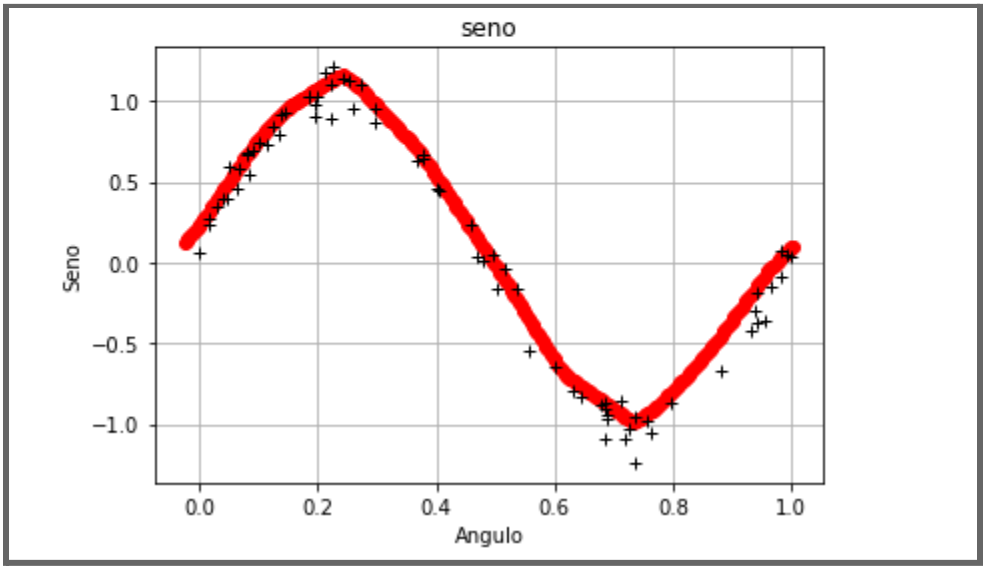


Figura 7(classificação modelo 3 padrão).

O parâmetro *callbacks* da função *fit* da biblioteca Keras possibilita a entrada de múltiplas opções, ele recebe uma lista. Foi adicionado a opção de carregamento por barra facilmente com o callback *TqdmCallback(verbose=2).* O resultado disso é apresentado na Figura 8. A barra parece estar *bugada* pelo notebook, porém a contagem funciona e ajuda a realizar os experimentos.

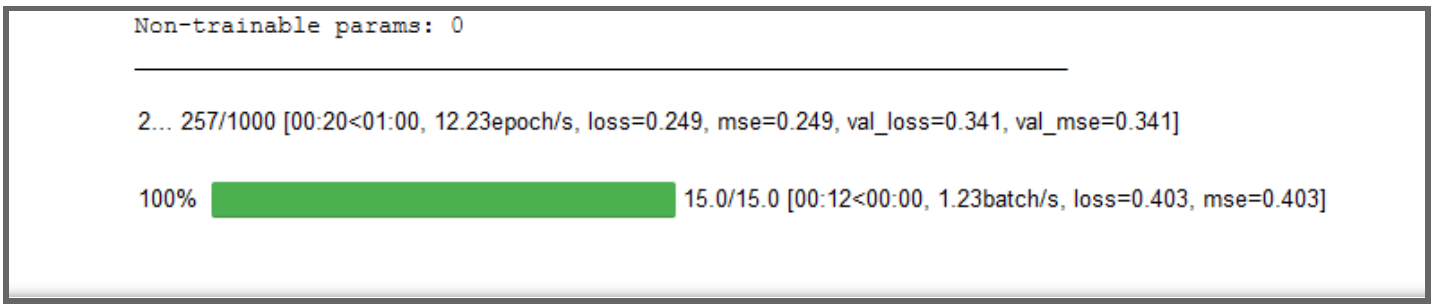


Figura 8.

Por acaso o código já veio com um número exagerado de epochs para o primeiro modelo, enquanto analisava o código ele acabou rodando por um bom tempo com esse valor exagerado de epochs, o resultado do histórico da função custo se da pela Figura 9. Provavelmente a partir do epoch 2500 é feito um overfitting completamente exagerado do conjunto de treinamento, chegando pelo visto muito perto de 0 o erro tanto no treino quanto na validação.

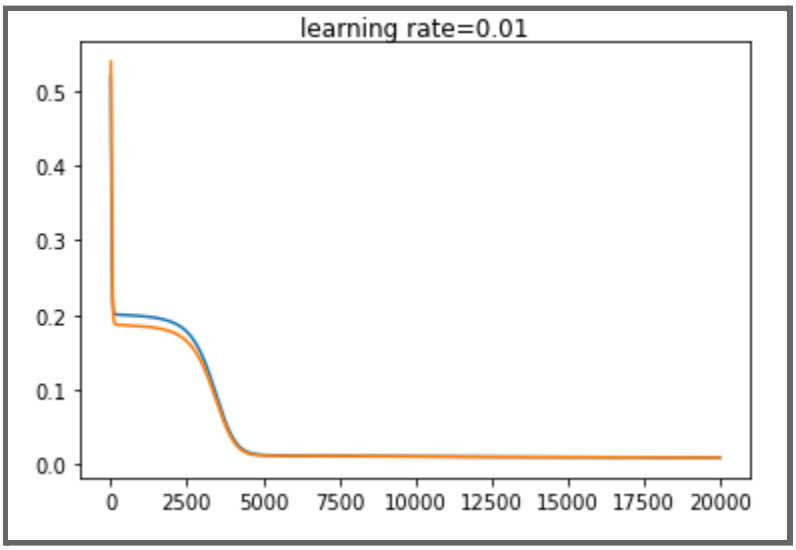


Figura 9.

1. **Aumentar o "ruído" dos dados de treinamento e ver como a rede se comporta com níveis de variância nos dados de treinamento;**

Podemos aumentar o ruído do dataset gerado alterando a linha *s=np.random.normal(0,0.1,size = (100,1))* no qual é responsável justamente por isso. Alteramos os valores de s para serem uma distribuição normal com centro em 0 e um desvio padrão um pouco normal do que o por padrão, agora de 0.35. A Figura 10 demonstra como a distribuição do dataset no formato seno se dá de maneira bem mais esparsa.

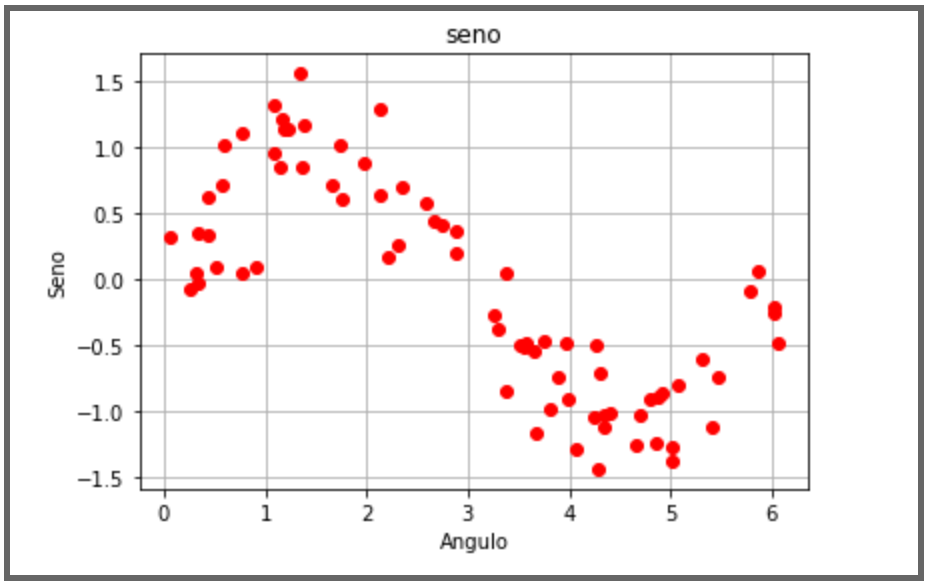


Figura 10 - Dataset ruidoso (std dev=0.35).

Nenhum dos modelos conseguiu obter um resultado satisfatório com o dataset ruidoso, nem com um numero maior de epochs de 500, só o modelo 3 pareceu demonstrar uma certa convergência com o conjunto ruidoso. Com uma alta divergência nos valores custo obtidos. Como monstram a Figura 11.

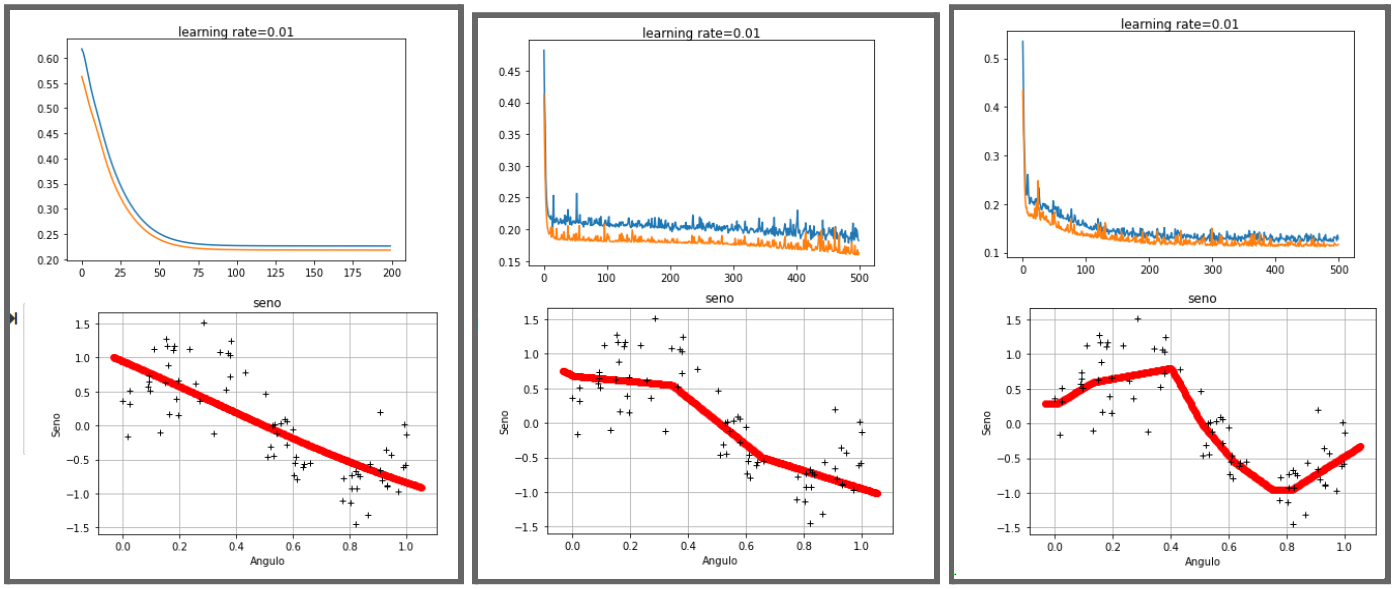


Figura 11 (dataset ruidoso - modelos 1, 2 e 3 em sequência).

**2. Aumentar e/ou diminuir a taxa de aprendizado;** Para isto, voltamos a o ruído do dataset para std dev=0.1, como por padrão. Rodando para uma parametrização de → **epochs=1000**, **learning rate=0.1**, momentum=0.8, **patience=250.** Não se pode obter uma convergência no aprendizado, o learning rate é muito alto e há muita divergência na redução do custo, no qual não parece estar convergindo para um valor mínimo. Nem com uma alta tolerância de paciecia e numero total de epochs não ajudaram a uma possível convergência com um learning rate tão alto.

Já um learning rate mais baixo de 0.001 se obteve uma resposta melhor em ambos modelos, pricipalemnte para o modelo 3.

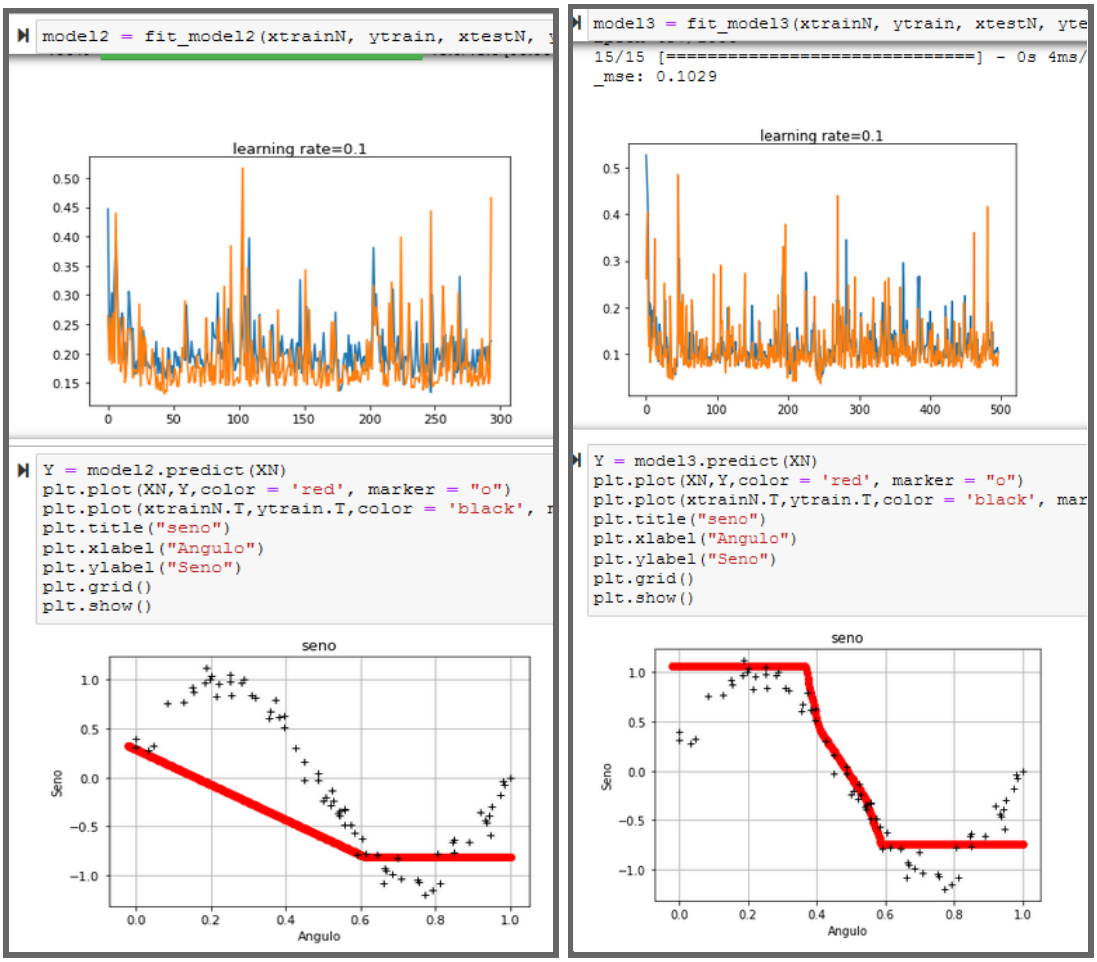


Figura 12 (learning rate **alto 0.1** - modelos 2 e 3).

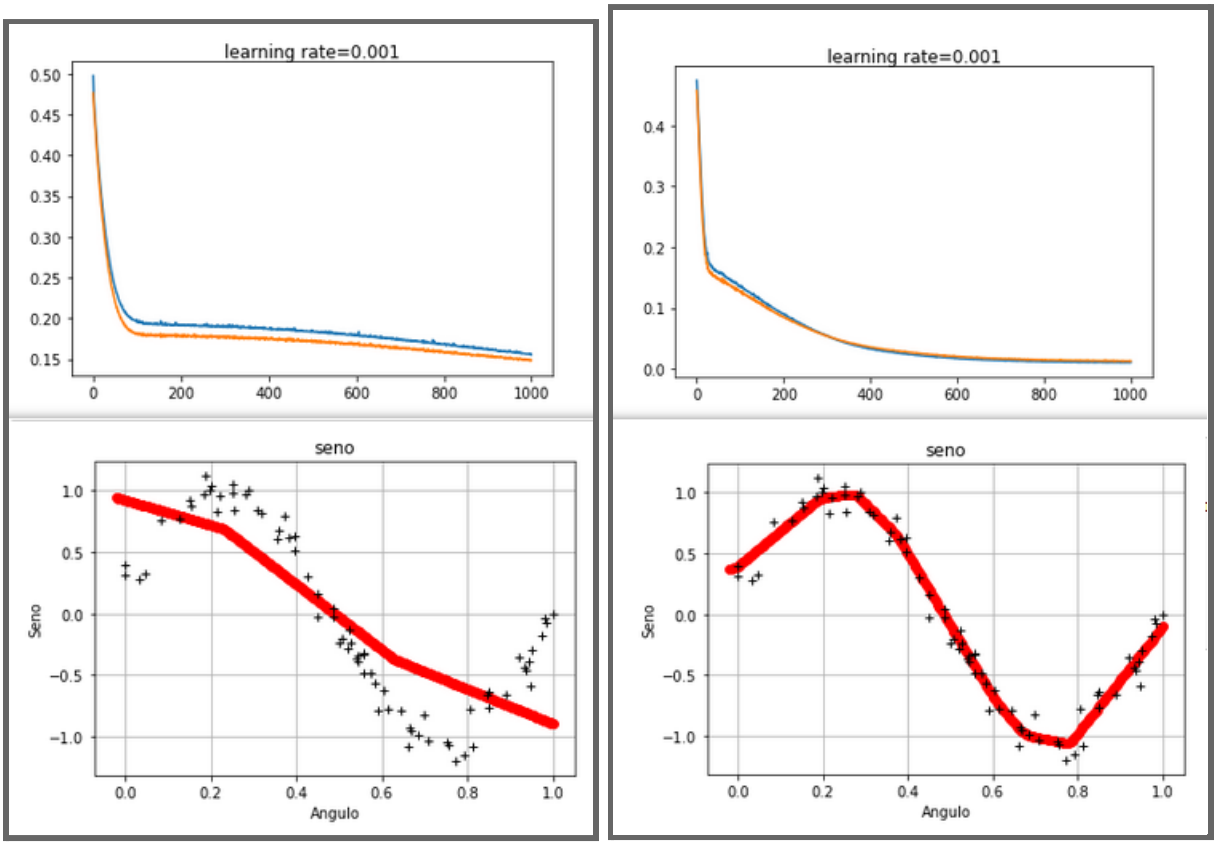


Figura 13 (learning rate **baixo 0.001** - modelos 2 e 3).

**3. Trocar as funções de ativação das camadas da rede neural.** A Figura 14 apresenta tentativas frustradas de remover o aspecto linear do primeiro modelo, a função de ativação da camada de saída foi removida. Só a classificação com função de ativação relu na camada interna apresentou um aspecto não linear, um comportamento parecido com o da própria função relu. A Figura 15 mostra como a função de ativação *softmax* deve ser usada somente para problemas de classificação. Nesta situação temos um problema de regressão, logo faz mais sentido a camada de saída ter função *linear*.

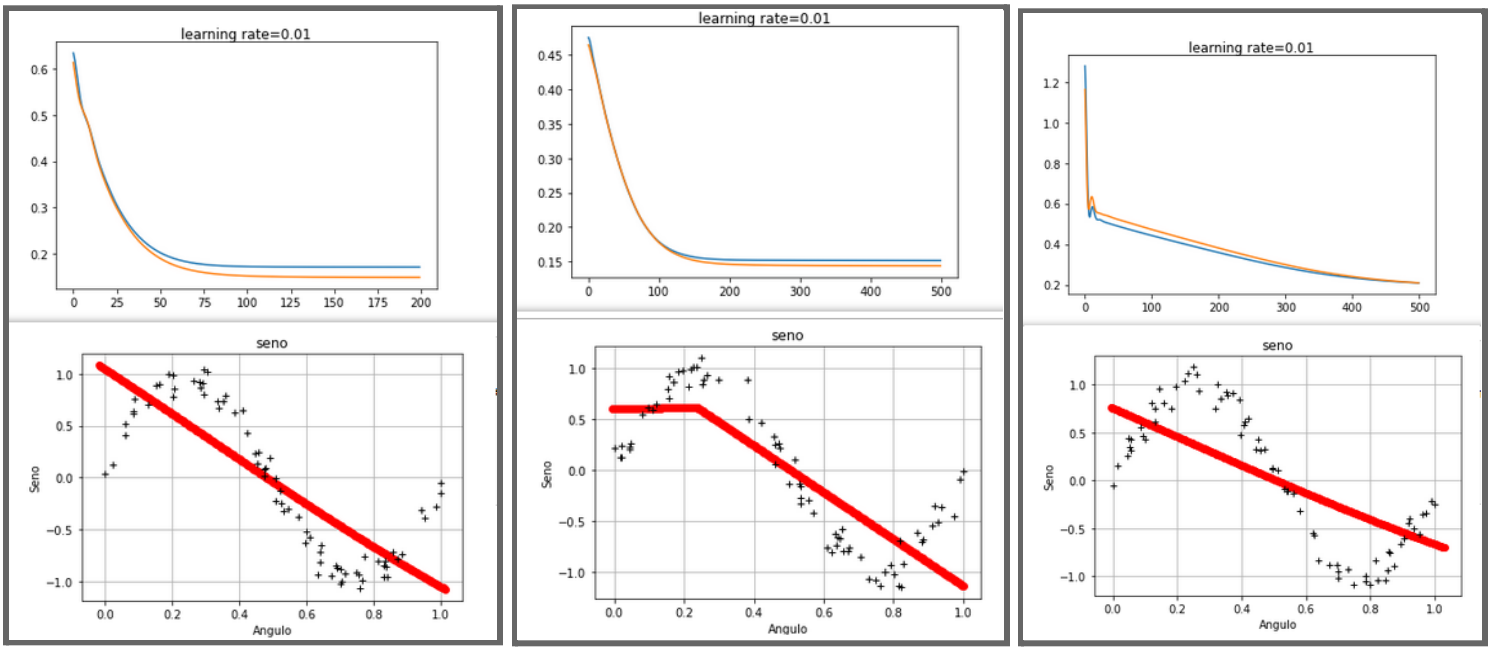


Figura 14 (modelo 1 - sem ativação na saída, camada escondida *tanh* seguido de *relu* e *sigmoid*)

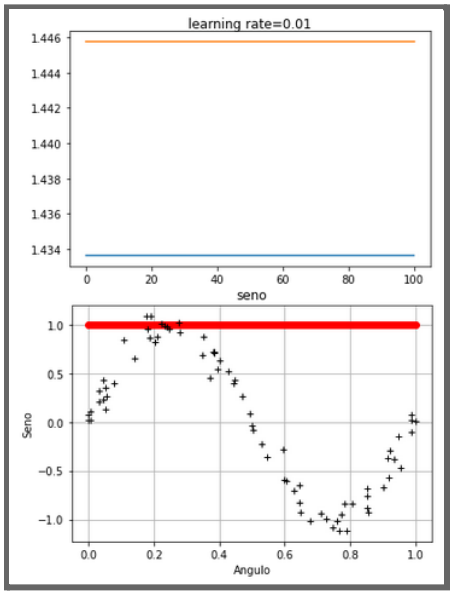


Figura 15 (modelo 1 com função *softmax* na camada de saída).

A Figura 16 mostra duas tentativas no modelo 2 de diferentes funções de ativação tanh e sigmoid respectivamente, ambas com função linear na camada de saída. Interessante como a função sigmoid está gerando uma classificação linear. Por outro lado a Figura 5 apresenta os valores originais e uma função relu na camada interna e função linear na saída consegue gerar um comportamento não linear, não entendi. A Figura 17 mostra o modelo 3 com função linear na saída em vez de vazio, a capacidade de classificação se mostrou bem parecida.

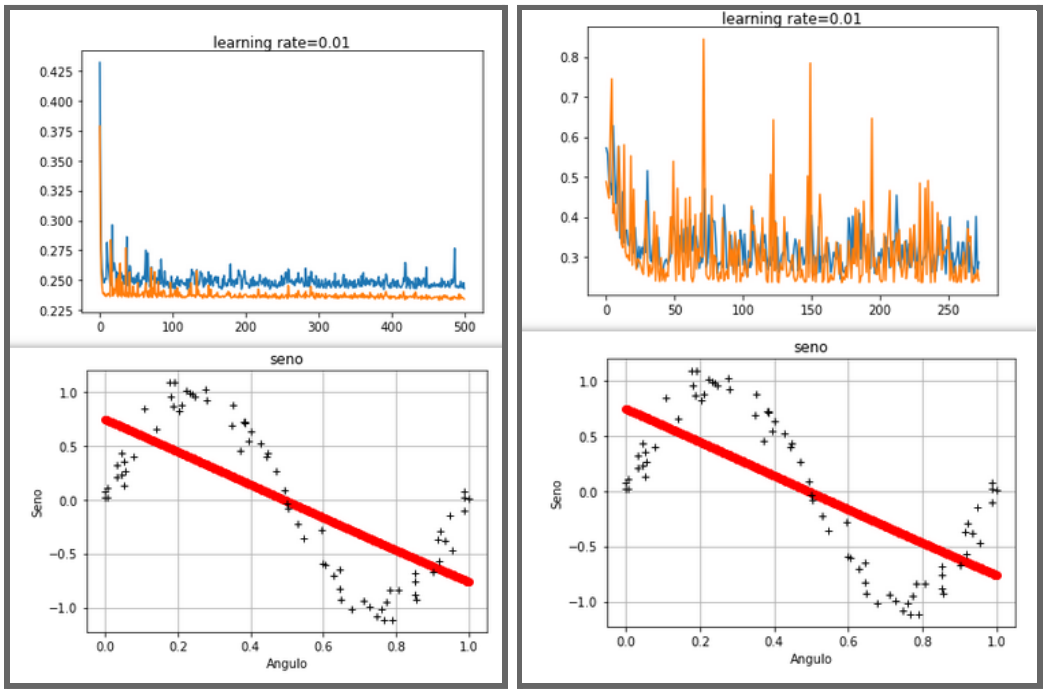


Figura 16 (modelo 2 com funções *tanh* e *sigmoid* respectivamente na camada interna e *linear* na saída).

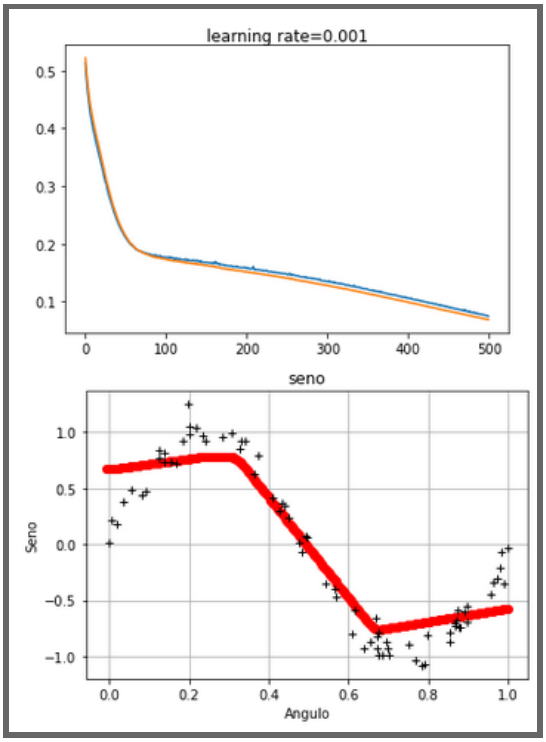


Figura 17(modelo 3 com ultima camada com função linear em vez de vazio).

As Figuras 18 e 19 demonstram alterações na primeira e segunda camada do modelo 3 a partir dos valores padrões (se alterou somente a função de uma camada escondida), com as funções tanh e sigmoid, e realmente parece que a melhor função de ativação para o problema em questão é a função **RELU**, uma vez que os valores padrões com duas camadas relus deram o melhor resultado de aprendizado.

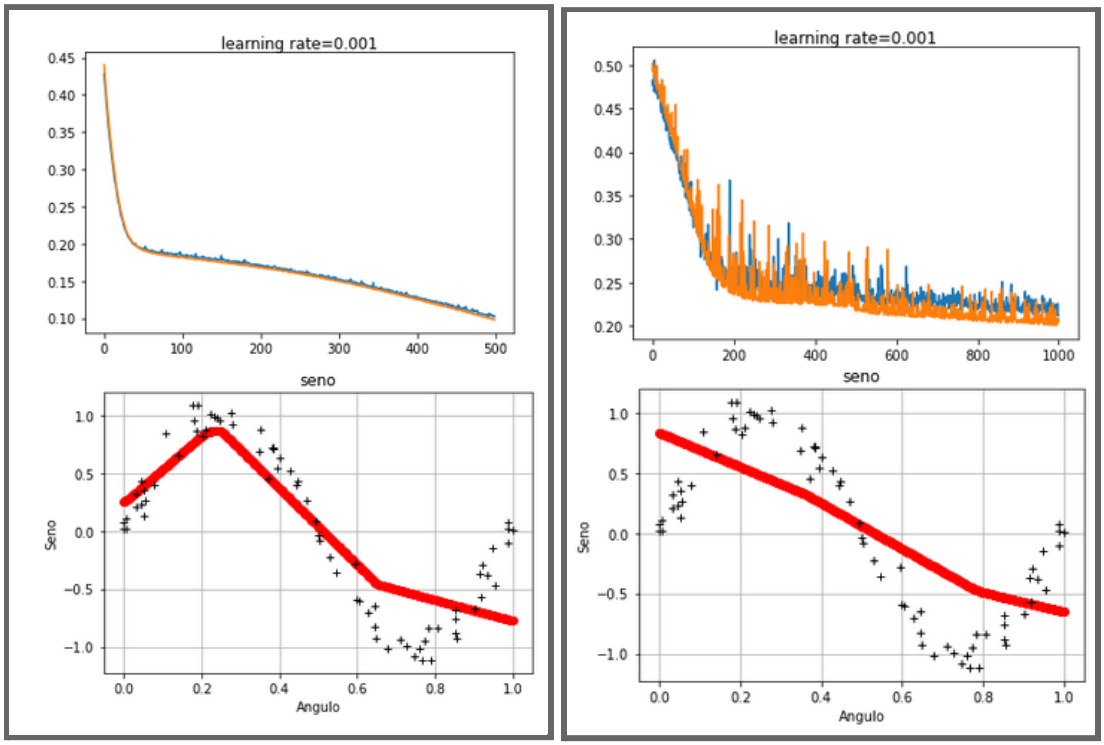


Figura 18 (alterações na função da **primeira** camada do modelo 3: *tanh* e *sigmoid*).

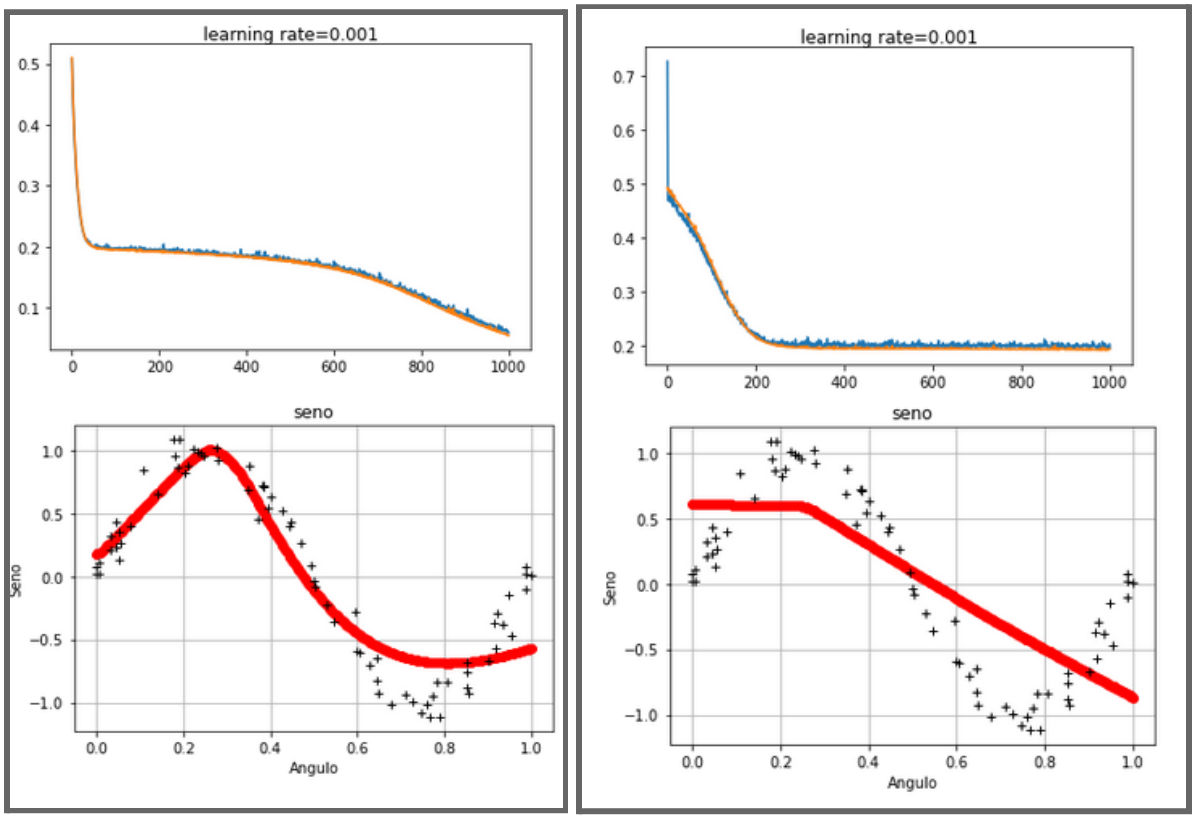


Figura 19 (alterações na função da **segunda** camada do modelo 3: *tanh* e *sigmoid*).

**4. Ver como a rede se comporta para prever o valor para entradas menores que 0 (zero) e maiores que 2*pi;** A figura 20 mostra como não funcionou uma primeira tentativa de utilizar as redes para prever valores maiores que 2*pi. A Figura 21 mostra que também se tentou estender o alcance do dataset de treinamento, porém os resultados de aprendizado não foram satisfatórios. A Figura 22 mostra que também não se obteve um aprendizado satisfatório com valores de entrada menores que 0. Entendo que para o modelo classifica corretamente as entradas modificadas, também se precisa modificar o dataset de treinamento, para que durante o treino o modelo consiga aprender as modificações. Porém tanto os dados a serem classificados, quanto o dataset de treinamento precisam ser modificados, acredito que algo ficou faltando, a Figura 23 mostra algumas das modificações tentadas no código para realizar este experimento.

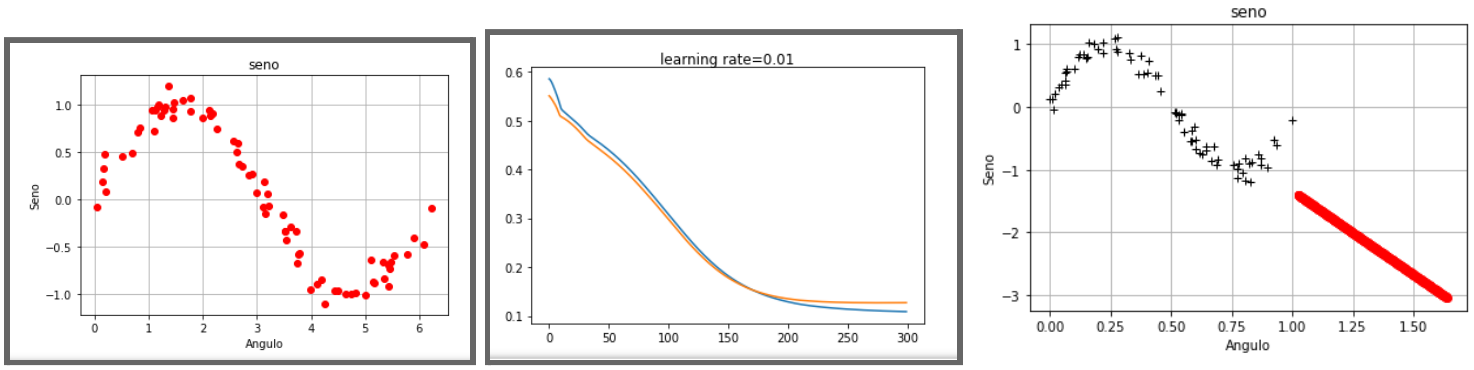


Figura 20 (dataset e modelo 1 com entradas **maiores que 2*pi**).

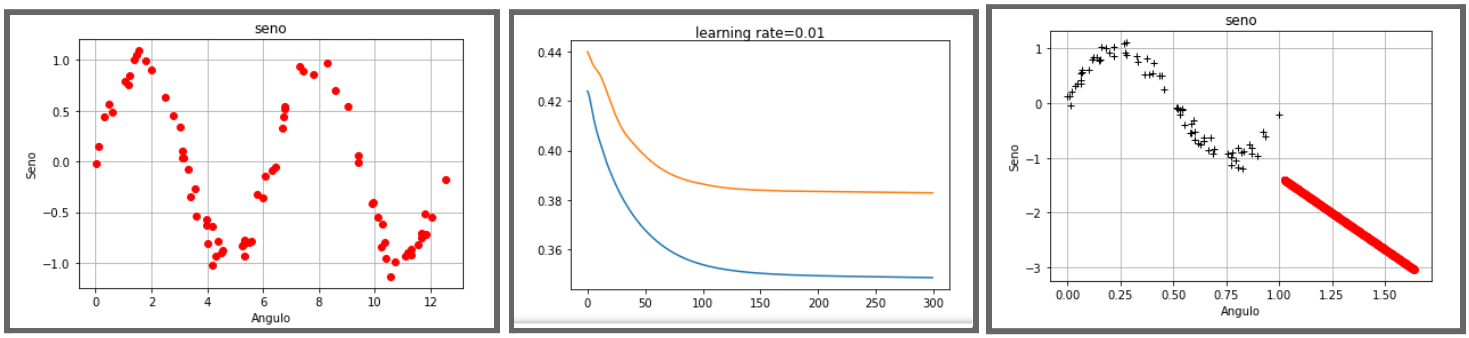


Figura 21 (dataset **estendido** e modelo 1 com entradas **maiores que 2*pi**).

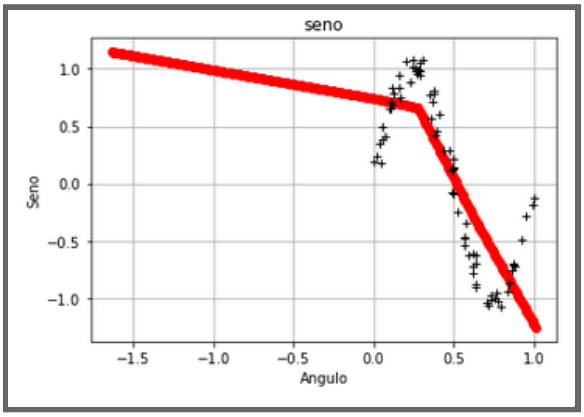


Figura 22 (valores de entrada menores que 0).

```
X = np.linspace(0.0 , 2.0 * np.pi, 360).reshape(-1, 1)
# X = np.linspace(2.0 * np.pi,10, 360).reshape(-1, 1)
# X = np.linspace(-10,2.0 * np.pi, 360).reshape(-1, 1)
# X = np.linspace(-1 ,1, 360).reshape(-1, 1)
print(X)
print("-----------------")
XN = scaler.transform(X)
print(XN)
```

```
    x = np.random.rand(ntot,1)*2*np.pi
#     x = -np.random.rand(ntot,1)*2*np.pi
#     x = np.random.rand(ntot,1)*4*np.pi
#     s = np.linspace(0.0 , 2.0 * np.pi, ntot).reshape(-1, 1)
    s=np.random.normal(0,0.1,size = (100,1))
    print(s.shape)
```

Figura 23 (tentativas de modificações no código).

6. **Aprender o problema inverso (trocar a entrada pelas saídas e vice-versa);** As Figuras 24 e 25 mostram o aprendizado da rede a partir de um dataset invertido, trocando-se X por Y.

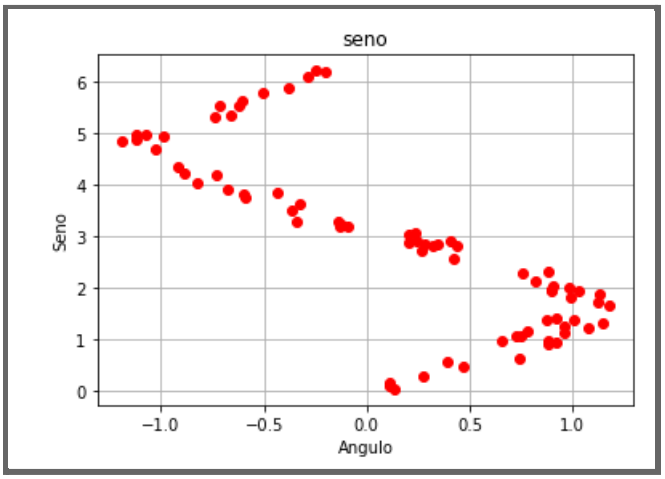


Figura 24 (dataset **invertido**).

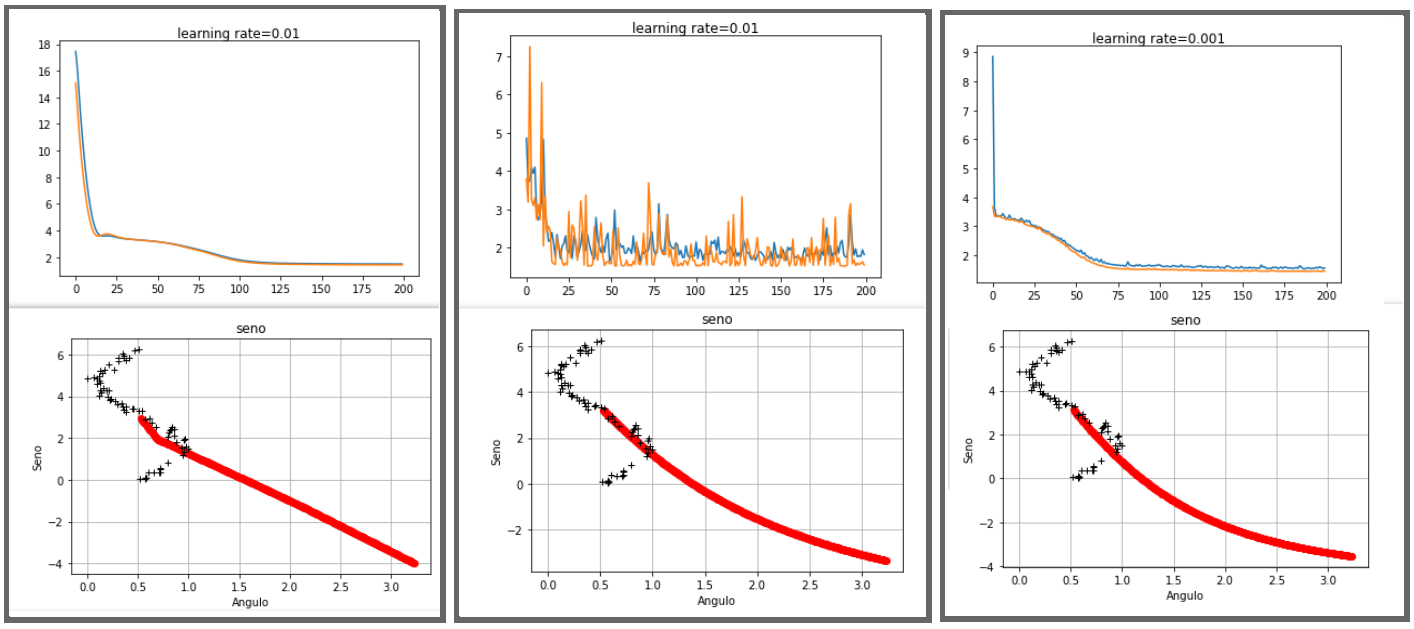


Figura 25.