MLP para regressão

Boston housing regression dataset
https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv
(https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv)
https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.names
(https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.names)
Previsão do valor das casas dadas caracteristicas da casa e vizinhança. Trata-se de prever um único valor númerico.

Imports

In [25]:
```
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torch.utils.data import random_split
import torch
from torch import Tensor
from torch.nn import Linear
from torch.nn import Sigmoid, ReLU
from torch.nn import Module
from torch.optim import SGD, Adam
from torch.nn import MSELoss
from torch.nn.init import xavier_uniform_
```

In [26]:
```
# Constants

#path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master
/housing.csv'
PATH = 'housing.csv'

# não estão a ser utilizados para já
device = torch.device("cpu") #torch.device("cuda" if torch.cuda.is_a
vailable() else "cpu")

EPOCHS = 200
BATCH_SIZE = 32
LEARNING_RATE = 0.001
```

1. Preparar os Dados

In [27]:
```
# definição classe para o dataset
class CSVDataset(Dataset):
    # ler o dataset
    def __init__(self, path):
        # ler o ficheiro csv para um dataframe
        df = pd.read_csv(path, header=None)
        # separar os inputs e os outputs
        self.X = df.values[:, :-1]
        self.y = df.values[:, -1]
        # garantir que os inputs sejam floats
        self.X = self.X.astype('float32')
        self.y = self.y.astype('float32')
        # garantir o shape correto para o label
        self.y = self.y.reshape((len(self.y), 1))

    # número de casos no dataset
    def __len__(self):
        return len(self.X)

    # retornar um caso
    def __getitem__(self, idx):
        return [self.X[idx], self.y[idx]]

    # retornar índices para casos de treino de de teste
    def get_splits(self, n_test=0.33):
        # calcular tamanho para o split
        test_size = round(n_test * len(self.X))
        train_size = len(self.X) - test_size
        # calcular o split do houldout
        return random_split(self, [train_size, test_size])#, generat
or=torch.Generator().manual_seed(42))

# preparar o dataset
def prepare_data(path):
    # criar uma instância do dataset
    dataset = CSVDataset(path)
    # calcular split
    train, test = dataset.get_splits()
    # preparar data loaders
    train_dl = DataLoader(train, batch_size=len(train), shuffle=Tru
e) #32 len(train)
    test_dl = DataLoader(test, batch_size=1024, shuffle=False)
    train_dl_all = DataLoader(train, batch_size=len(train), shuffle=
False)
    test_dl_all = DataLoader(test, batch_size=len(test), shuffle=Fal
se)
    return train_dl, test_dl, train_dl_all, test_dl_all

# preparar os dados
train_dl, test_dl,  train_dl_all, test_dl_all = prepare_data(PATH)
```

2. Definir o Modelo

In [28]:

```
from torchinfo import summary

# Definição classe para o modelo
class MLP(Module):
    # definir elementos do modelo
    def __init__(self, n_inputs):
        super(MLP, self).__init__()
        # input para a primeira camada
        self.hidden1 = Linear(n_inputs, 40)
        xavier_uniform_(self.hidden1.weight)
        self.act1 = ReLU()
        # segunda camada
        self.hidden2 = Linear(40, 18)
        xavier_uniform_(self.hidden2.weight)
        self.act2 = ReLU()
        self.hidden3 = Linear(10, 18)
        xavier_uniform_(self.hidden2.weight)
        self.act3 = ReLU()
        # terceira camada e output
        self.hidden4 = Linear(18, 1) # um nodo para o output do valo
r previsto
        xavier_uniform_(self.hidden3.weight) # Glorot initialization
        # neste caso não colocamos função ativação no final para con
siderar uma ativação linear

    # sequência de propagação do input
    def forward(self, X):
        # input para a primeira camada
        X = self.hidden1(X)
        X = self.act1(X)
        # segunda camada
        X = self.hidden2(X)
        X = self.act2(X)
        #X = self.hidden3(X)
        #X = self.act3(X)
        # terceira camada e output
        X = self.hidden4(X)
        return X

# definir a rede neuronal
model = MLP(13)
# visualizar a rede
print(summary(model, input_size=(BATCH_SIZE, 13), verbose=0)) #verbo
se=2 Show weight and bias layers in full detail
model.to(device)
```

```
==========================================================================
========================
Layer (type:depth-idx)                   Output Shape
Param #
==========================================================================
========================
├─Linear: 1-1                            [32, 40]
560
├─ReLU: 1-2                              [32, 40]
--
├─Linear: 1-3                            [32, 18]
738
├─ReLU: 1-4                              [32, 18]
--
├─Linear: 1-5                            [32, 1]
19
==========================================================================
========================
Total params: 1,317
Trainable params: 1,317
Non-trainable params: 0
Total mult-adds (M): 0.04
==========================================================================
========================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.02
Params size (MB): 0.01
Estimated Total Size (MB): 0.02
==========================================================================
========================
```

Out[28]:
```
MLP(
  (hidden1): Linear(in_features=13, out_features=40, bias=True)
  (act1): ReLU()
  (hidden2): Linear(in_features=40, out_features=18, bias=True)
  (act2): ReLU()
  (hidden3): Linear(in_features=10, out_features=18, bias=True)
  (act3): ReLU()
  (hidden4): Linear(in_features=18, out_features=1, bias=True)
)
```

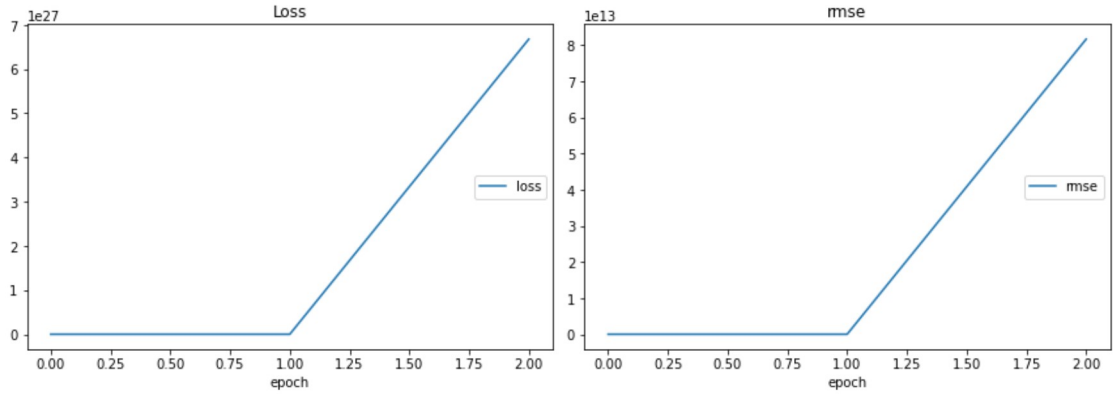
3. Treinar o Modelo

In [29]:
```
#versão com sgd
from livelossplot import PlotLosses

# treino do modelo
def train_model(train_dl, model):
    liveloss = PlotLosses() ##para visualizarmos o processo de trein
o
    # definir o loss e a função de otimização
    criterion = MSELoss() # neste caso implementa a mean squared err
or
    optimizer = SGD(model.parameters(), lr=LEARNING_RATE, momentum=
0.9) # stochastic gradient descent
    #optimizer = Adam(model.parameters(), lr=LEARNING_RATE)
    # iterar as epochs
    for epoch in range(EPOCHS):
        logs = {} ##para visualizarmos o processo de treino
        # iterar as batchs
        epoch_loss  = 0 ##para visualizarmos o processo de treino
        epoch_rmse  = 0 ##para visualizarmos o processo de treino
        for i, (inputs, labels) in enumerate(train_dl): # backpropag
ation
            # inicializar os gradientes
            optimizer.zero_grad() #coloca os gradientes de todos os
parâmetros a zero
            # calcular o output do modelo
            outputs = model(inputs)
            # calcular o loss
            loss = criterion(outputs, labels)#.unsqueeze(1))
            rmse = np.sqrt(mean_squared_error(labels.reshape((len(la
bels), 1)).numpy(), outputs.detach().numpy()))
            # atribuição alteraçoes "In the backward pass we receive
a Tensor containing the gradient of the loss
            # with respect to the output, and we need to compute the
gradient of the loss with respect to the input.
            loss.backward()
            # update pesos do modelo
            optimizer.step()
            #s ó para multiclass:
            #valores, predictions = torch.max(outputs, 1) #retorna u
m tensor com os indices do valor maximo em cada caso
            epoch_loss += loss.item()
            epoch_rmse += rmse.item()

        print(f'Epoch {epoch:03}: | Loss: {epoch_loss/len(train_d
l):.5f} | RMSE: {epoch_rmse/len(train_dl):.3f}')
        logs['loss'] = epoch_loss/len(train_dl) ##para visualizarmos
o processo de treino
        logs['rmse'] = epoch_rmse/len(train_dl) ##para visualizarmos
o processo de treino
        liveloss.update(logs) ##para visualizarmos o processo de tre
ino
        liveloss.send() ##para visualizarmos o processo de treino

# treinar o modelo
train_model(train_dl, model)
```




```
Loss
        loss                    (min:   94.228, max:      inf, cu
r:      inf)
rmse
        rmse                    (min:    9.707, max:      inf, cu
r:      inf)
```

```
---------------------------------------------------------------------
---------
ValueError                                Traceback (most recent c
all last)
<ipython-input-29-427b62fcc61a> in <module>
     41
     42 # treinar o modelo
---> 43 train_model(train_dl, model)

<ipython-input-29-427b62fcc61a> in train_model(train_dl, model)
     23             # calcular o loss
     24             loss = criterion(outputs, labels)#.unsqueeze
(1))
---> 25             rmse = np.sqrt(mean_squared_error(labels.resha
pe((len(labels), 1)).numpy(), outputs.detach().numpy()))
     26             # atribuição alteraçoes "In the backward pass
we receive a Tensor containing the gradient of the loss
     27             #with respect to the output, and we need to co
mpute the gradient of the loss with respect to the input.

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py
in inner_f(*args, **kwargs)
     70                                 FutureWarning)
     71         kwargs.update({k: arg for k, arg in zip(sig.parame
ters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74

/opt/conda/lib/python3.7/site-packages/sklearn/metrics/_regressio
n.py in mean_squared_error(y_true, y_pred, sample_weight, multiout
put, squared)
    254     """
    255     y_type, y_true, y_pred, multioutput = _check_reg_targe
ts(
--> 256         y_true, y_pred, multioutput)
    257     check_consistent_length(y_true, y_pred, sample_weight)
    258     output_errors = np.average((y_true - y_pred) ** 2, axi
s=0,

/opt/conda/lib/python3.7/site-packages/sklearn/metrics/_regressio
n.py in _check_reg_targets(y_true, y_pred, multioutput, dtype)
```

Mudar o *learning rate* para 0.00000001

Imports

In [30]:
```
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torch.utils.data import random_split
import torch
from torch import Tensor
from torch.nn import Linear
from torch.nn import Sigmoid, ReLU
from torch.nn import Module
from torch.optim import SGD, Adam
from torch.nn import MSELoss
from torch.nn.init import xavier_uniform_
```

In [31]:
```
# Constants

#path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master
/housing.csv'
PATH = 'housing.csv'

# não estão a ser utilizados para já
device = torch.device("cpu") #torch.device("cuda" if torch.cuda.is_a
vailable() else "cpu")

EPOCHS = 200
BATCH_SIZE = 32
LEARNING_RATE = 0.00000001
```

1. Preparar os Dados

In [32]:
```
# definição classe para o dataset
class CSVDataset(Dataset):
    # ler o dataset
    def __init__(self, path):
        # ler o ficheiro csv para um dataframe
        df = pd.read_csv(path, header=None)
        # separar os inputs e os outputs
        self.X = df.values[:, :-1]
        self.y = df.values[:, -1]
        # garantir que os inputs sejam floats
        self.X = self.X.astype('float32')
        self.y = self.y.astype('float32')
        # garantir o shape correto para o label
        self.y = self.y.reshape((len(self.y), 1))

    # número de casos no dataset
    def __len__(self):
        return len(self.X)

    # retornar um caso
    def __getitem__(self, idx):
        return [self.X[idx], self.y[idx]]

    # retornar índices para casos de treino de de teste
    def get_splits(self, n_test=0.33):
        # calcular tamanho para o split
        test_size = round(n_test * len(self.X))
        train_size = len(self.X) - test_size
        # calcular o split do houldout
        return random_split(self, [train_size, test_size])#, generat
or=torch.Generator().manual_seed(42))

# preparar o dataset
def prepare_data(path):
    # criar uma instância do dataset
    dataset = CSVDataset(path)
    # calcular split
    train, test = dataset.get_splits()
    # preparar data loaders
    train_dl = DataLoader(train, batch_size=len(train), shuffle=Tru
e) #32 len(train)
    test_dl = DataLoader(test, batch_size=1024, shuffle=False)
    train_dl_all = DataLoader(train, batch_size=len(train), shuffle=
False)
    test_dl_all = DataLoader(test, batch_size=len(test), shuffle=Fal
se)
    return train_dl, test_dl, train_dl_all, test_dl_all

# preparar os dados
train_dl, test_dl,  train_dl_all, test_dl_all = prepare_data(PATH)
```

2. Definir o Modelo

In [33]:

```
from torchinfo import summary

# Definição classe para o modelo
class MLP(Module):
    # definir elementos do modelo
    def __init__(self, n_inputs):
        super(MLP, self).__init__()
        # input para a primeira camada
        self.hidden1 = Linear(n_inputs, 40)
        xavier_uniform_(self.hidden1.weight)
        self.act1 = ReLU()
        # segunda camada
        self.hidden2 = Linear(40, 18)
        xavier_uniform_(self.hidden2.weight)
        self.act2 = ReLU()
        self.hidden3 = Linear(10, 18)
        xavier_uniform_(self.hidden2.weight)
        self.act3 = ReLU()
        # terceira camada e output
        self.hidden4 = Linear(18, 1) # um nodo para o output do valo
r previsto
        xavier_uniform_(self.hidden3.weight) # Glorot initialization
        # neste caso não colocamos função ativação no final para con
siderar uma ativação linear

    # sequência de propagação do input
    def forward(self, X):
        # input para a primeira camada
        X = self.hidden1(X)
        X = self.act1(X)
         # segunda camada
        X = self.hidden2(X)
        X = self.act2(X)
        #X = self.hidden3(X)
        #X = self.act3(X)
        # terceira camada e output
        X = self.hidden4(X)
        return X

# definir a rede neuronal
model = MLP(13)
# visualizar a rede
print(summary(model, input_size=(BATCH_SIZE, 13), verbose=0)) #verbo
se=2 Show weight and bias layers in full detail
model.to(device)
```

```
==========================================================================
========================
Layer (type:depth-idx)                   Output Shape
Param #
==========================================================================
========================
├─Linear: 1-1                            [32, 40]
560
├─ReLU: 1-2                              [32, 40]
--
├─Linear: 1-3                            [32, 18]
738
├─ReLU: 1-4                              [32, 18]
--
├─Linear: 1-5                            [32, 1]
19
==========================================================================
========================
Total params: 1,317
Trainable params: 1,317
Non-trainable params: 0
Total mult-adds (M): 0.04
==========================================================================
========================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.02
Params size (MB): 0.01
Estimated Total Size (MB): 0.02
==========================================================================
========================
```

Out[33]:
```
MLP(
  (hidden1): Linear(in_features=13, out_features=40, bias=True)
  (act1): ReLU()
  (hidden2): Linear(in_features=40, out_features=18, bias=True)
  (act2): ReLU()
  (hidden3): Linear(in_features=10, out_features=18, bias=True)
  (act3): ReLU()
  (hidden4): Linear(in_features=18, out_features=1, bias=True)
)
```

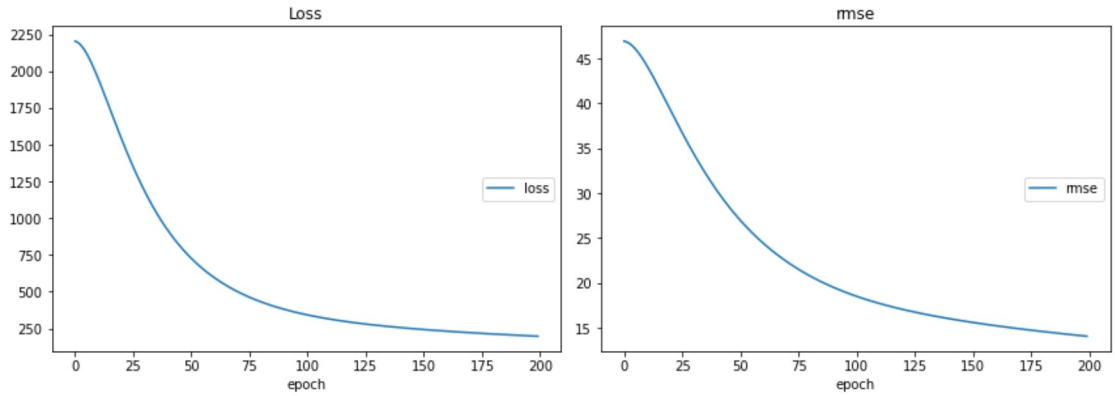
3. Treinar o Modelo

In [34]:
```
#versão com sgd
from livelossplot import PlotLosses

# treino do modelo
def train_model(train_dl, model):
    liveloss = PlotLosses() ##para visualizarmos o processo de trein
o
    # definir o loss e a função de otimização
    criterion = MSELoss() # neste caso implementa a mean squared err
or
    optimizer = SGD(model.parameters(), lr=0.00000001, momentum=0.9)
# stochastic gradient descent
    #optimizer = Adam(model.parameters(), lr=LEARNING_RATE)
    # iterar as epochs
    for epoch in range(EPOCHS):
        logs = {} ##para visualizarmos o processo de treino
        # iterar as batchs
        epoch_loss  = 0 ##para visualizarmos o processo de treino
        epoch_rmse  = 0 ##para visualizarmos o processo de treino
        for i, (inputs, labels) in enumerate(train_dl): # backpropag
ation
            # inicializar os gradientes
            optimizer.zero_grad() # coloca os gradientes de todos os
parâmetros a zero
            # calcular o output do modelo
            outputs = model(inputs)
            # calcular o loss
            loss = criterion(outputs, labels)#.unsqueeze(1))
            rmse = np.sqrt(mean_squared_error(labels.reshape((len(la
bels), 1)).numpy(), outputs.detach().numpy()))
            # atribuição alteraçoes "In the backward pass we receive
a Tensor containing the gradient of the loss
            # with respect to the output, and we need to compute the
gradient of the loss with respect to the input.
            loss.backward()
            # update pesos do modelo
            optimizer.step()
            # só para multiclass:
            #valores, predictions = torch.max(outputs, 1) #retorna u
m tensor com os índices do valor máximo em cada caso
            epoch_loss += loss.item()
            epoch_rmse += rmse.item()

        print(f'Epoch {epoch:03}: | Loss: {epoch_loss/len(train_d
l):.5f} | RMSE: {epoch_rmse/len(train_dl):.3f}')
        logs['loss'] = epoch_loss/len(train_dl) ##para visualizarmos
o processo de treino
        logs['rmse'] = epoch_rmse/len(train_dl) ##para visualizarmos
o processo de treino
        liveloss.update(logs) ##para visualizarmos o processo de tre
ino
        liveloss.send() ##para visualizarmos o processo de treino

# treinar o modelo
train_model(train_dl, model)
```




```
Loss
        loss                      (min:   94.228, max:      inf, cu
r:  196.776)
rmse
        rmse                      (min:    9.707, max:      inf, cu
r:   14.028)
```

4. Avaliar o Modelo

In [35]:

```
def evaluate_model(test_dl, model):
    predictions = list()
    actual_values = list()
    for i, (inputs, labels) in enumerate(test_dl):
        # avaliar o modelo com os casos de teste
        yprev = model(inputs)
        # retirar o array numpy
        yprev = yprev.detach().numpy()
        actual = labels.numpy()
        actual = actual.reshape((len(actual), 1))
        # guardar
        predictions.append(yprev)
        actual_values.append(actual)
        break
    predictions, actual_values = np.vstack(predictions), np.vstack(a
ctual_values)
    return actual_values, predictions

# avaliar o modelo
actual_values, predictions = evaluate_model(test_dl, model)
#actuals, predictions = evaluate_model(train_dl, model)

for r,p in zip(actual_values, predictions):
    print(f'real:{r} previsão:{p}')

# calcular a accuracy
mse = mean_squared_error(actual_values, predictions)
print(f'MSE: {mse:0.3f}, RMSE: {np.sqrt(mse):0.3f}\n')
```

real:[20.8] previsão:[26.320744]
real:[21.7] previsão:[12.321862]
real:[19.1] previsão:[25.636793]
real:[50.] previsão:[13.30473]
real:[12.3] previsão:[24.80757]
real:[13.4] previsão:[33.332172]
real:[19.1] previsão:[18.694496]
real:[25.3] previsão:[11.317577]
real:[23.9] previsão:[11.379128]
real:[21.1] previsão:[11.507861]
real:[10.4] previsão:[27.208082]
real:[20.] previsão:[18.377962]
real:[18.5] previsão:[16.022297]
real:[30.1] previsão:[3.9519322]
real:[20.6] previsão:[13.694244]
real:[11.9] previsão:[16.320364]
real:[31.6] previsão:[4.5174265]
real:[25.] previsão:[11.887]
real:[18.9] previsão:[15.796472]
real:[21.9] previsão:[26.367434]
real:[24.1] previsão:[7.269646]
real:[19.2] previsão:[20.470457]
real:[27.9] previsão:[26.999815]
real:[30.8] previsão:[5.7649727]
real:[32.7] previsão:[10.99944]
real:[21.] previsão:[15.1222515]
real:[31.1] previsão:[6.338579]
real:[19.6] previsão:[18.496515]
real:[24.] previsão:[13.745564]
real:[15.6] previsão:[19.88939]
real:[21.7] previsão:[13.5599785]
real:[19.4] previsão:[18.984825]
real:[50.] previsão:[4.0270653]
real:[20.3] previsão:[16.392826]
real:[22.6] previsão:[15.550161]
real:[50.] previsão:[14.600843]
real:[7.] previsão:[28.59653]
real:[23.8] previsão:[14.891795]
real:[13.5] previsão:[32.972874]
real:[22.1] previsão:[14.8443985]
real:[45.4] previsão:[12.959691]
real:[19.5] previsão:[19.071295]
real:[28.4] previsão:[14.443004]
real:[24.3] previsão:[15.515378]
real:[27.] previsão:[18.092209]
real:[5.] previsão:[24.437567]
real:[36.5] previsão:[14.746069]
real:[17.2] previsão:[15.626017]
real:[22.8] previsão:[12.903787]
real:[20.8] previsão:[26.43465]
real:[21.2] previsão:[19.51406]
real:[19.1] previsão:[24.45323]
real:[20.6] previsão:[7.429851]
real:[24.5] previsão:[6.0709887]
real:[26.6] previsão:[14.8357115]
real:[23.] previsão:[24.128878]
real:[19.6] previsão:[12.057953]
real:[24.1] previsão:[12.333398]
real:[16.1] previsão:[25.334446]
real:[20.7] previsão:[13.637028]

real:[50.] previsão:[19.326733]
real:[23.6] previsão:[16.802368]
real:[14.3] previsão:[20.023294]
real:[34.9] previsão:[8.482582]
real:[13.4] previsão:[19.90082]
real:[18.2] previsão:[16.680508]
real:[19.8] previsão:[16.788416]
real:[21.6] previsão:[16.541822]
real:[22.] previsão:[16.05221]
real:[14.8] previsão:[17.057673]
real:[10.4] previsão:[28.316027]
real:[18.9] previsão:[15.14467]
real:[32.5] previsão:[16.968084]
real:[14.9] previsão:[33.312008]
real:[17.2] previsão:[32.25013]
real:[19.4] previsão:[17.031742]
real:[15.3] previsão:[17.747404]
real:[17.8] previsão:[26.083223]
real:[16.4] previsão:[26.391905]
real:[22.9] previsão:[11.815135]
real:[19.3] previsão:[16.699377]
real:[8.8] previsão:[27.11236]
real:[10.5] previsão:[24.551596]
real:[13.6] previsão:[18.048454]
real:[17.5] previsão:[18.010166]
real:[20.7] previsão:[12.820136]
real:[20.3] previsão:[14.844646]
real:[48.3] previsão:[15.940145]
real:[21.9] previsão:[6.6062226]
real:[16.1] previsão:[13.501952]
real:[15.] previsão:[25.988577]
real:[20.3] previsão:[14.469474]
real:[33.] previsão:[13.225347]
real:[22.] previsão:[16.871237]
real:[48.5] previsão:[4.0746965]
real:[21.4] previsão:[25.669577]
real:[15.6] previsão:[16.960213]
real:[12.5] previsão:[25.109116]
real:[32.2] previsão:[4.773971]
real:[15.6] previsão:[17.606813]
real:[18.5] previsão:[13.69967]
real:[23.4] previsão:[10.3379755]
real:[50.] previsão:[16.86012]
real:[33.4] previsão:[12.641144]
real:[22.6] previsão:[25.0655]
real:[10.9] previsão:[30.31033]
real:[26.7] previsão:[17.234552]
real:[27.1] previsão:[16.119856]
real:[9.5] previsão:[31.77515]
real:[32.9] previsão:[8.526074]
real:[24.4] previsão:[12.7316]
real:[8.5] previsão:[25.319609]
real:[46.] previsão:[11.907378]
real:[17.8] previsão:[18.351166]
real:[23.1] previsão:[16.98885]
real:[24.4] previsão:[14.682659]
real:[15.2] previsão:[25.207876]
real:[16.5] previsão:[18.317657]
real:[29.] previsão:[15.216823]
real:[24.7] previsão:[13.420566]

```
real:[17.1] previsão:[9.517304]
real:[13.2] previsão:[13.745051]
real:[20.3] previsão:[16.200544]
real:[50.] previsão:[27.245089]
real:[23.9] previsão:[12.124791]
real:[20.] previsão:[17.425491]
real:[21.7] previsão:[17.361614]
real:[23.1] previsão:[28.750212]
real:[32.4] previsão:[9.403607]
real:[24.1] previsão:[15.401263]
real:[22.4] previsão:[14.543431]
real:[19.3] previsão:[18.593897]
real:[5.6] previsão:[24.72878]
real:[19.9] previsão:[15.271207]
real:[13.4] previsão:[29.054197]
real:[44.] previsão:[4.787198]
real:[17.2] previsão:[25.69124]
real:[20.4] previsão:[15.631738]
real:[24.4] previsão:[14.526257]
real:[11.9] previsão:[25.6391]
real:[41.3] previsão:[19.339193]
real:[28.7] previsão:[15.531961]
real:[50.] previsão:[27.087305]
real:[28.5] previsão:[5.6466317]
real:[16.6] previsão:[16.274733]
real:[21.2] previsão:[13.451267]
real:[24.8] previsão:[9.212874]
real:[43.5] previsão:[12.445947]
real:[50.] previsão:[26.5323]
real:[7.5] previsão:[32.44164]
real:[46.7] previsão:[13.39709]
real:[15.2] previsão:[17.534805]
real:[23.1] previsão:[7.3383517]
real:[36.4] previsão:[12.900008]
real:[29.] previsão:[8.80026]
real:[20.5] previsão:[14.113303]
real:[22.2] previsão:[11.254841]
real:[12.] previsão:[25.722805]
real:[20.2] previsão:[13.314038]
real:[23.8] previsão:[16.177809]
real:[23.1] previsão:[15.804984]
real:[17.1] previsão:[25.44525]
real:[8.5] previsão:[25.477224]
real:[20.8] previsão:[16.761356]
real:[15.6] previsão:[16.897638]
real:[24.8] previsão:[7.780828]
real:[19.3] previsão:[16.783903]
MSE: 241.983, RMSE: 15.556
```

5. Usar o Modelo

In [36]:
```
# fazer uma previsão utilizando um caso
def predict(row, model):
    # converter row para tensor
    row = Tensor([row])
    # fazer a previsão
    yprev = model(row)
    # retirar o array numpy
    yprev = yprev.detach().numpy()
    return yprev


# fazer uma única previsão (classe esperada=1)
row = [0.00632,18.00,2.310,0,0.5380,6.5750,65.20,4.0900,1,296.0,15.3
0,396.90,4.98]
yprev = predict(row, model)
[[yprev]]=yprev
print(yprev)
print(f'Predicted: {yprev:.3f}')
```

```
13.745565
Predicted: 13.746
```

In [ ]: