PREVISÃO DE SÉRIES TEMPORAIS COM MACHINE LEARNING

Discente: Thaís Medeiros

IMPORTAÇÕES

```
import numpy as np
import matplotlib.pyplot as plt
import random
import pandas as pd

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader

from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
```

DEFINIÇÃO DA SEED E SELEÇÃO DO DISPOSITIVO DE TREINO


```
SEED = 42

random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.mps.manual_seed(SEED)

device = torch.device("mps"
                      if torch.backends.mps.is_available()
                      else "cpu")
print("Device:", device)
```

SÉRIE TEMPORAL

```python
df = pd.read_csv('co2_emissions_processed.csv')

# Converter coluna year para índice
df["year"] = pd.to_datetime(df["year"],
format="%Y")
df = df.set_index("year")

# Série
serie = df["total_ghg"].values
```

kaggle

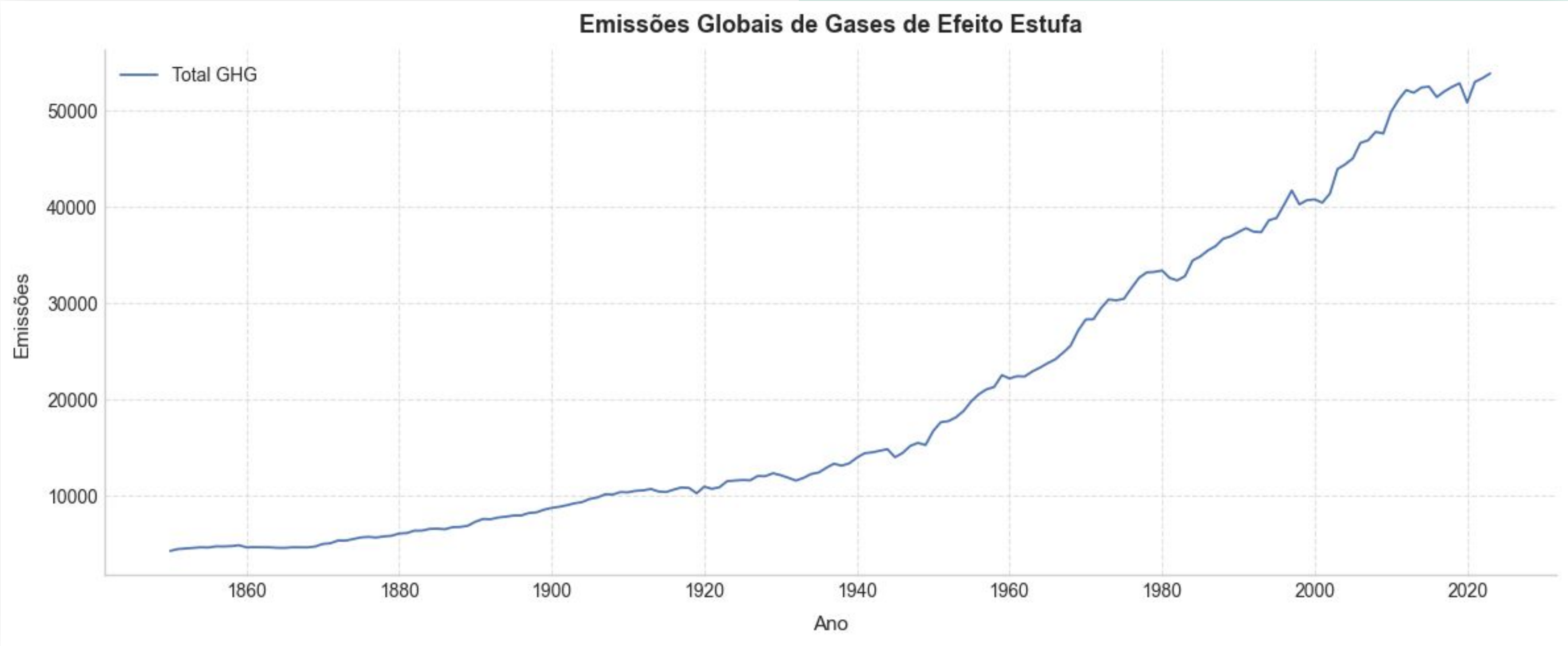
ULRIK THYGE PEDERSEN · UPDATED 3 YEARS AGO

CO2 Emissions

Can you forecast CO2 Emissions?

174 dados

SÉRIE TEMPORAL



JANELA DESLIZANTE, TREINO E TESTE

```
# Janela
time_steps=3

# Split 80%/20%
train_size = int(len(serie) * 0.8)
serie_train = serie[:train_size]
serie_test  = serie[train_size:]

print("Tamanho treino:",
len(serie_train))
print("Tamanho teste :", len(serie_test))

# Tamanho treino: 139
# Tamanho teste : 35
```

ESCALA E CONJUNTOS

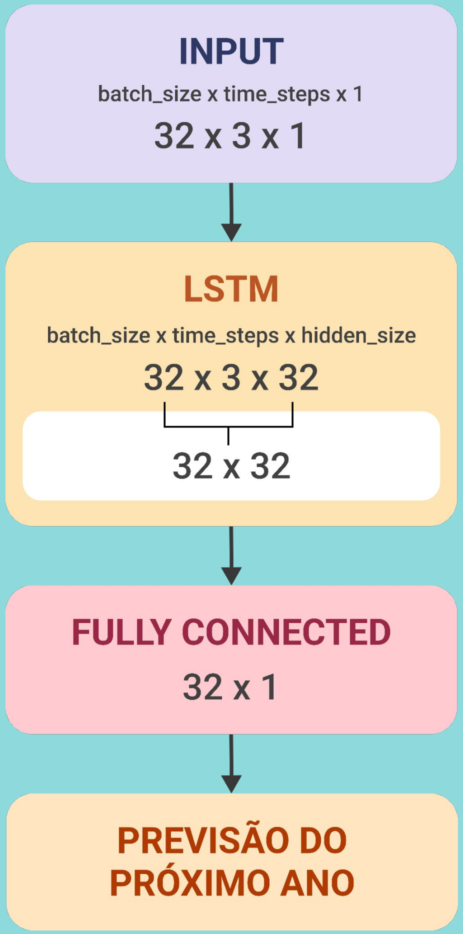
```
# Escala
scaler = StandardScaler()
serie_train_scaled = scaler.fit_transform(serie_train.reshape(-1,
1)).flatten()
serie_test_scaled  = scaler.transform(serie_test.reshape(-1, 1)).flatten()

def create_dataset(series, time_steps=time_steps):
    X, y = [], []
    for i in range(len(series) - time_steps):
        X.append(series[i:i+time_steps])
        y.append(series[i+time_steps])
    return np.array(X), np.array(y)

# Conjuntos (X,y)
X_train, y_train = create_dataset(serie_train_scaled, time_steps)
X_test,  y_test  = create_dataset(serie_test_scaled,  time_steps)

print("X_train shape:", X_train.shape)  # [N_train, time_steps] (136, 3)
print("X_test shape :", X_test.shape)   # [N_test, time_steps] (32, 3)

print("y_train shape:", y_train.shape)  # [N_train,] (136,)
print("y_test shape :", y_test.shape)   # [N_test,] (32,)
```

PREPARAÇÃO DE DADOS NO PYTORCH

```python
class TimeSeriesDataset(Dataset):
    def __init__(self, X, y):
        # X: [N, time_steps], y: [N]
        # LSTM espera [batch, seq_len, features]
        self.X = torch.tensor(X, dtype=torch.float32).unsqueeze(-1)  # -> [N, T, 1]
        self.y = torch.tensor(y, dtype=torch.float32).unsqueeze(-1)  # -> [N, 1]

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

train_dataset = TimeSeriesDataset(X_train, y_train)
test_dataset  = TimeSeriesDataset(X_test,  y_test)

batch_size = 32
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=False)
test_loader  = DataLoader(test_dataset,  batch_size=batch_size, shuffle=False)

for xb, yb in train_loader:
    print("Batch X shape:", xb.shape)  # Batch X shape: torch.Size([32, 3, 1])
    print("Batch y shape:", yb.shape)  # Batch y shape: torch.Size([32, 1])
```

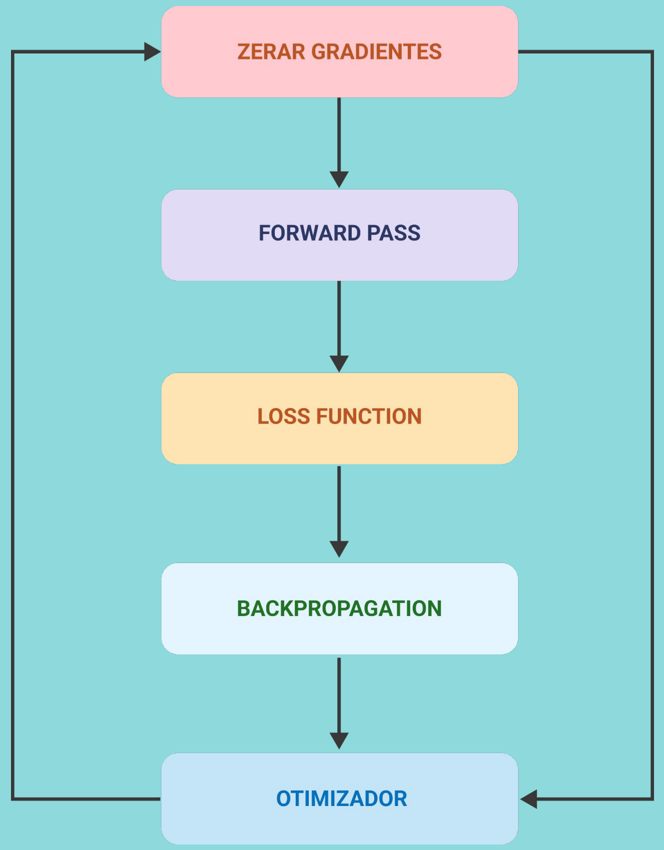
LSTM

```python
class LSTMRegressor(nn.Module):
    def __init__(self, input_size=1, hidden_size=20, num_layers=1):
        super().__init__()
        # batch_first=True -> entrada [batch, seq, feature]
        self.lstm = nn.LSTM(
            input_size=input_size,
            hidden_size=hidden_size,
            num_layers=num_layers,
            batch_first=True
        )
        self.fc = nn.Linear(hidden_size, 1)

    def forward(self, x):
        # x: [batch, seq_len, input_size]
        output, (h_n, c_n) = self.lstm(x)
        # output: [batch, seq_len, hidden_size]
        last_output = output[:, -1, :]  # pega o último passo de tempo
        out = self.fc(last_output)      # [batch, 1]
        return out

model = LSTMRegressor(input_size=1, hidden_size=32,
num_layers=1).to(device)
print(model)
```



TREINO

```
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-
3)
num_epochs = 1000

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0

    for X_batch, y_batch in train_loader:
        X_batch = X_batch.to(device)
        y_batch = y_batch.to(device)

        optimizer.zero_grad()
        y_pred = model(X_batch)
        loss = criterion(y_pred, y_batch)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * X_batch.size(0)

    epoch_loss = running_loss / len(train_dataset)

    if (epoch + 1) % 100 == 0:
        print(f"Epoch {epoch+1} - Loss
{epoch_loss:.6f}")
```

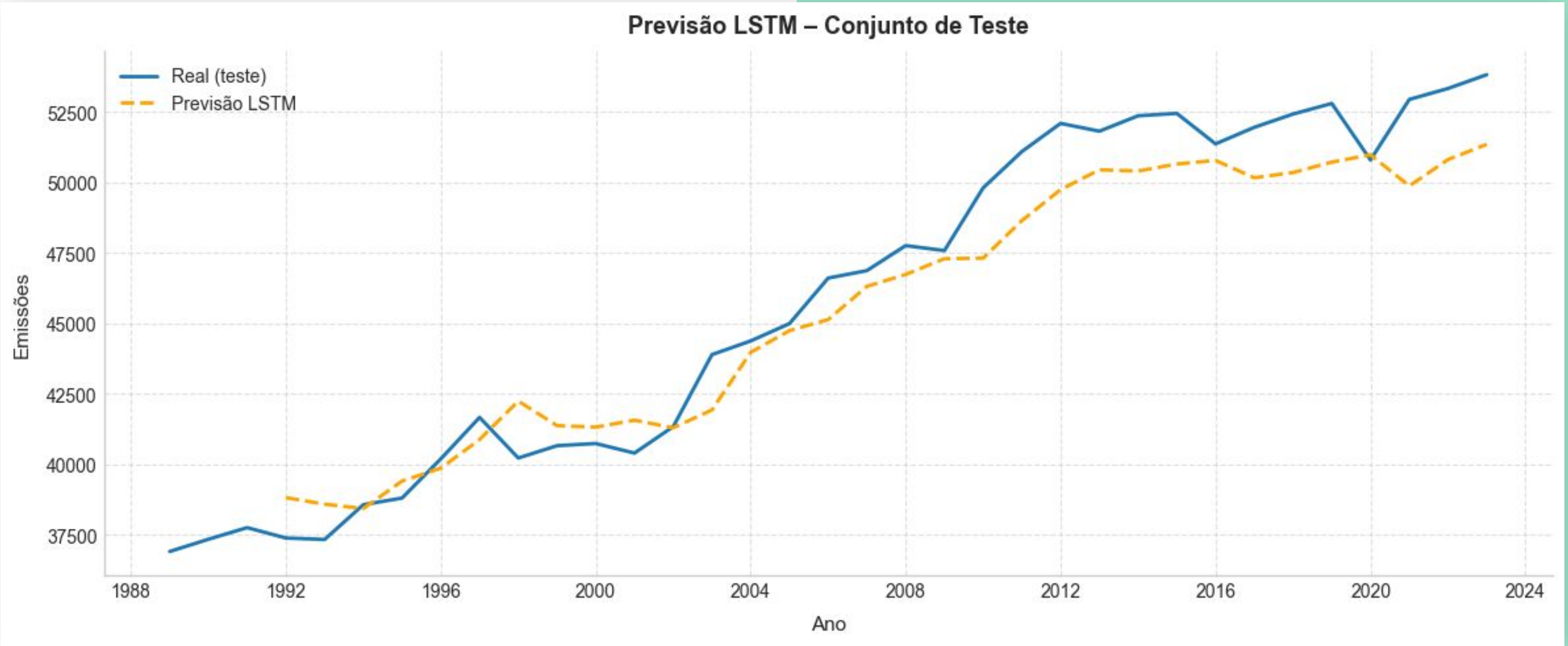


ÉPOCAS
1000

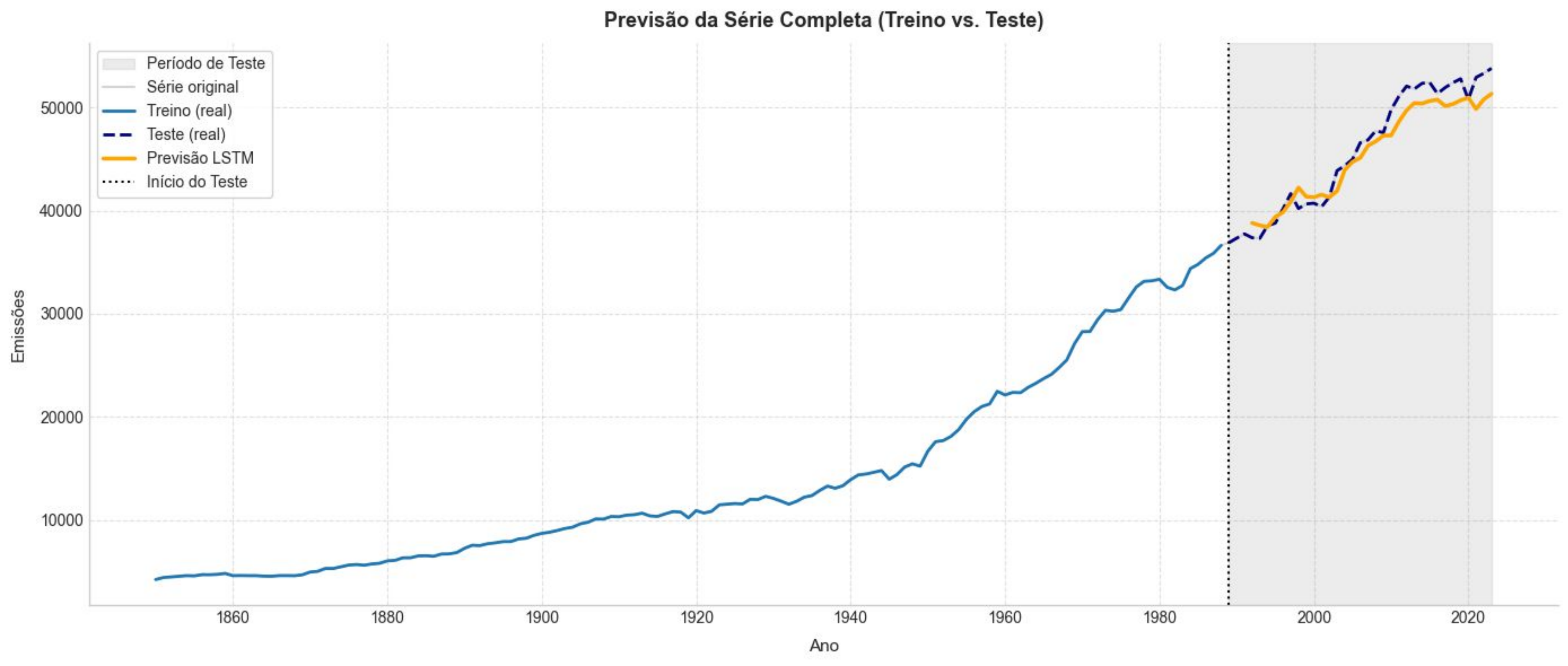
FUNÇÃO DE PERDA
MSE

OTIMIZADOR
ADAM

PREVISÃO

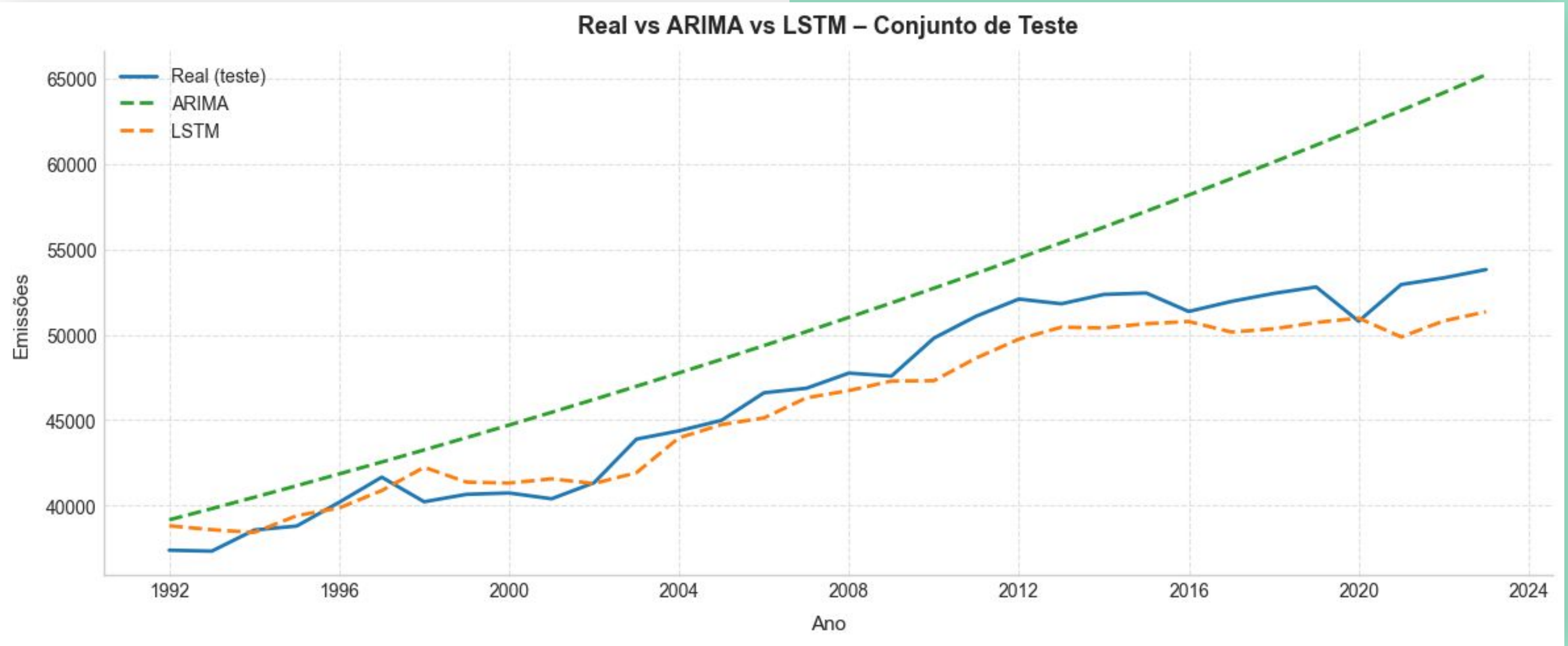

Previsão LSTM – Conjunto de Teste
Real (teste)
Previsão LSTM
52500
50000
47500
45000
42500
40000
37500
Emissões
1988
1992
1996
2000
2004
2008
2012
2016
2020
2024
Ano

PREVISÃO



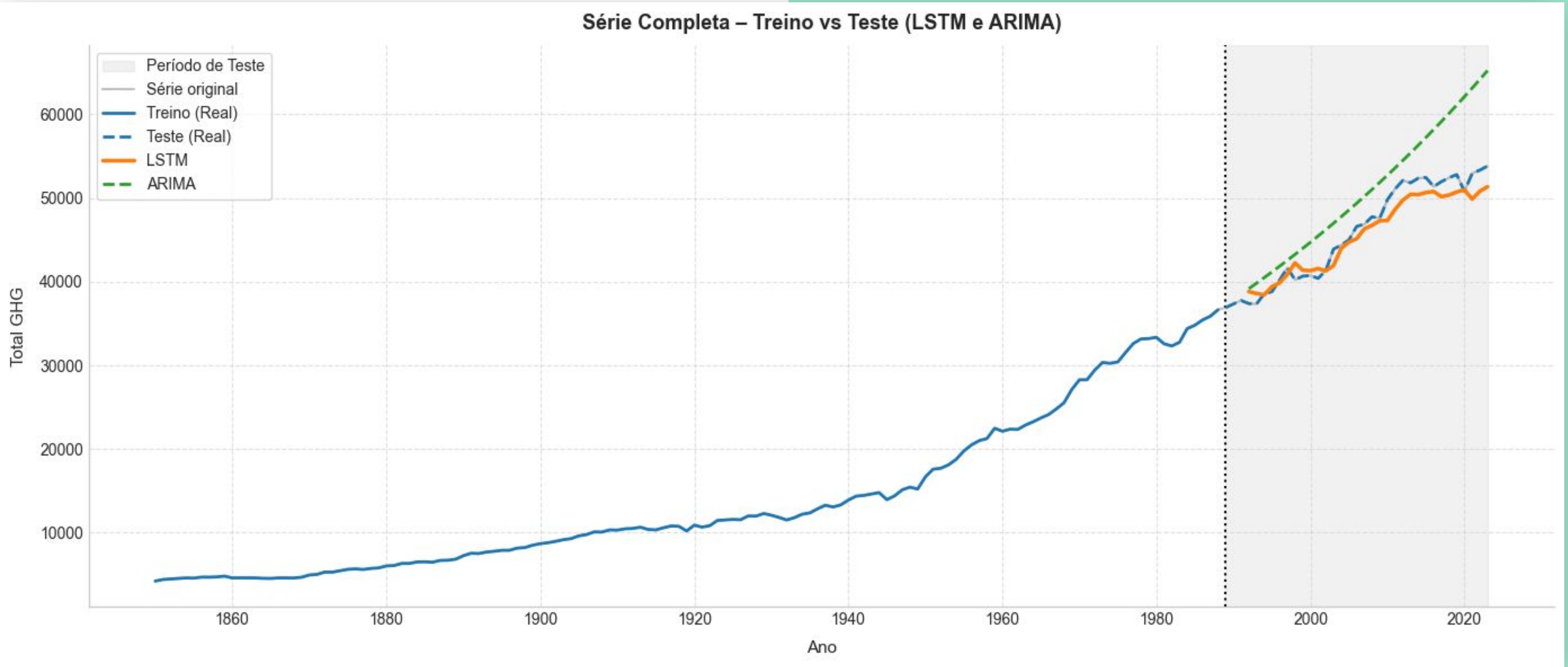
Previsão da Série Completa (Treino vs. Teste)
Período de Teste
Série original
Treino (real)
Teste (real)
Previsão LSTM
Início do Teste
50000
40000
30000
20000
10000
Emissões
1860
1880
1900
1920
1940
1960
1980
2000
2020
Ano

COMPARAÇÃO



Real vs ARIMA vs LSTM – Conjunto de Teste
Real (teste)
ARIMA
LSTM
65000
60000
55000
50000
45000
40000
Emissões
1992
1996
2000
2004
2008
2012
2016
2020
2024
Ano

COMPARAÇÃO



Série Completa – Treino vs Teste (LSTM e ARIMA)
Período de Teste
Série original
Treino (Real)
Teste (Real)
LSTM
ARIMA
60000
50000
40000
30000
20000
10000
Total GHG
1860
1880
1900
1920
1940
1960
1980
2000
2020
Ano

COMPARAÇÃO

Modelo	MSE	RMSE	MAE	R^2	n
ARIMA	3.02×10^7	5497,856	4655,346	0,022424	32
LSTM	2.48×10^6	1576,131	1319,546	0,919657	32

Métrica	Melhoria / Redução
MSE	-91,78%
RMSE	-71,33%
MAE	-71,66%

Obrigada

thais.araujo.707@ufrn.edu.br